HTML BASICS

HTML stands for Hyper Text Mark-up Language, which is the most widely used language on Web to develop web pages.

HTML elements are the building blocks of HTML pages

HTML elements are represented by tags

HTML tags are element names surrounded by angle brackets.

Read on to know more…

An HTML tag is as shown below:

```
<tagname>content goes here...</tagname>
```

The first tag in a pair is the **start tag,** the second tag is the **end tag**

The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

The start tag is also called the **opening tag**, and the end tag the **closing tag**.

HTML Attributes

Attributes provide **additional information** about an element

Attributes are always specified in **the start tag**

Attributes usually come in name/value pairs like: **name="value"**

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.

Paragraph Tag

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag

Line Break Tag

Whenever you use the **
** 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.

Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these 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.

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 −

In cases, where you do not want the client browser to break text, you should use a nonbreaking space entity ** ** 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 −

HTML Entities

Some characters are reserved in HTML.

If you use the less than (<) or greater than (>) signs in your text, the browser might mix them with tags.

Character entities are used to display reserved characters in HTML.

A character entity looks like this:

&*entity_name*;

OR

&#*entity_number*;

Entity names are case sensitive.

Result	Description	Entity Name	Entity Number
	non-breaking space		

<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark (apostrophe)	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

HTML Comment Tags

You can add comments to your HTML source by using the following syntax:

<!—this is a comment -->

HTML Formatting

Bold Text

Anything that appears within **...** element, is displayed in bold

Italic Text

Anything that appears within **...** element is displayed in italicized

Underlined Text

Anything that appears within **<u>...</u>** element

Superscript Text

The content of a **^{...}** element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

Subscript Text

The content of a **_{...}** element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

Write HTML Using Notepad/Notepad++/Any Text Editor

Write the following program in any text editor like Notepad++

```
<html>

	<head>

		<title>MY FIRST HTML PAGE</title>

	</head>

	<body bgcolor="red" text="white">

	<!--this is a comment-->

		<h1 align="center">SAMPLE HEADING</h1>

		<h2 align="center">SAMPLE HEADING</h2>

		<h3 align="center">SAMPLE HEADING</h3>
```

```
            <h4 align="center">SAMPLE HEADING</h4>

            <h5 align="center">SAMPLE HEADING</h5>

            <h6 align="center">SAMPLE HEADING</h6>

            <p>

                  TEST PARAGRAPH.....<br>

                  <!--HTML ENTITY:NON BREAKING SPACE-->

                  IN                  LINE                  2                  OF
<b><u><i>TEST      PARAGRAPH</i></u
></b>

            </p>

            <h1>&copy;</h1>

            <h1>&reg;</h1>

            <pre>

                  IN         LINE 1

                  IN LINE 2

            </pre>

      </body>

</html>
```

In Notepad++ go to Run option in the menu->Launch in Chrome.

This will open the chrome browser and give the following output:

Inserting Images

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.

For example:

```
<img src="butterfly.jpg" width="200px" height="200px"
                alt="cannot display image">
```

The alt attribute:

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

If a browser cannot find an image, it will display the value of the alt attribute:

Images in Another Folder

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/flower.jpg" width="200px" height="200px"
        alt="cannot display image">
```

HTML Links – Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

In HTML, links are defined with the **<a>** tag:

link text

The **href** attribute specifies the destination address (https://www.w3schools.com/html/) of the link.

The **link text** is the visible part (Visit our HTML tutorial).

Clicking on the link text will send you to the specified address.

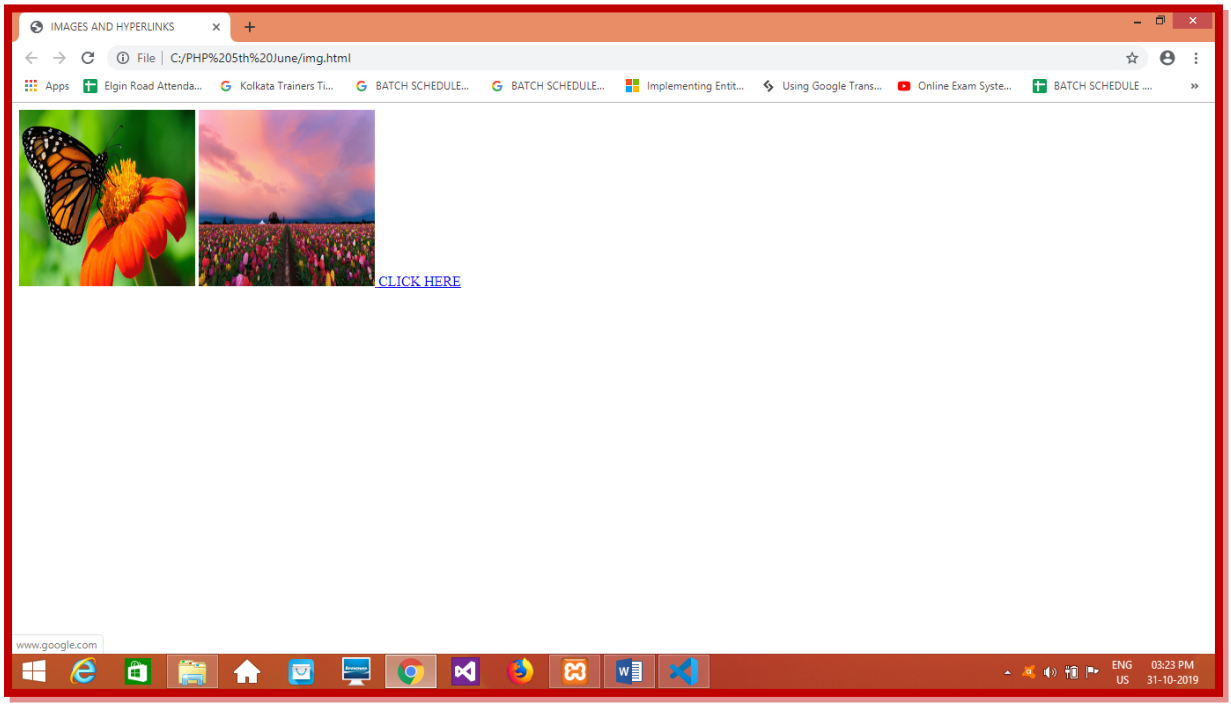
```
<a href="index.html">GO TO PAGE</a>
```

```
<a href="http://www.google.com">CLICK HERE</a>
```

```
<html>
    <head>
        <title>IMAGES & HYPERLINK</title>
    </head>
    <body>
        <img src="images/butterfly.jpg" width="200px" height="200px"
        alt="cannot display image">

        <br>
        <!--a anchor tag with an attribute href which is hyper reference-->

        <a href="http://www.google.com">
            <img src="images/flower.jpg" width="200px"
height="200px"
```

```
            alt="cannot display image">
            </a>
            <br>
            <a href="http://www.google.com">CLICK HERE</a>
        </body>
</html>
```

Sample output:



HTML Tables

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. The elements under <td> are regular and left aligned by default

```
<html>
        <head>
            <title>TABLES IN HTML</title>
        </head>
        <body>
```

```
            <table border="1px" align="center" cellpadding="25px"
cellspacing="0px"
            bgcolor="cyan" background="">
                  <tr bgcolor="green">
                        <th rowspan="2">ID</th>
                        <th rowspan="2" bgcolor="yellow">NAME</th>
                        <th colspan="3" bgcolor="red">SALARY</th>
                  </tr>
                  <tr>
                        <th>BASIC</th>
                        <th>DA</th>
                        <th>HRA</th>
                  </tr>
                  <tr>
                        <td>E001</td>
                        <td>AAAA</td>
                        <td>1234</td>
                        <td>126</td>
                        <td>934</td>
                  </tr>
                  <tr>
                        <td>E002</td>
                        <td>BBBB</td>
                        <td>2345</td>
                        <td>321</td>
                        <td>834</td>
                  </tr>
            </table>
      </body>
</html>
```

Here, the **border** is an attribute of <table> tag and it is used to put a border across all the cells. If you do not need a border, then you can use border = "0".

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. Headings, which are defined in <th> tag are centered and bold by default.

Cellpadding and Cellspacing Attributes

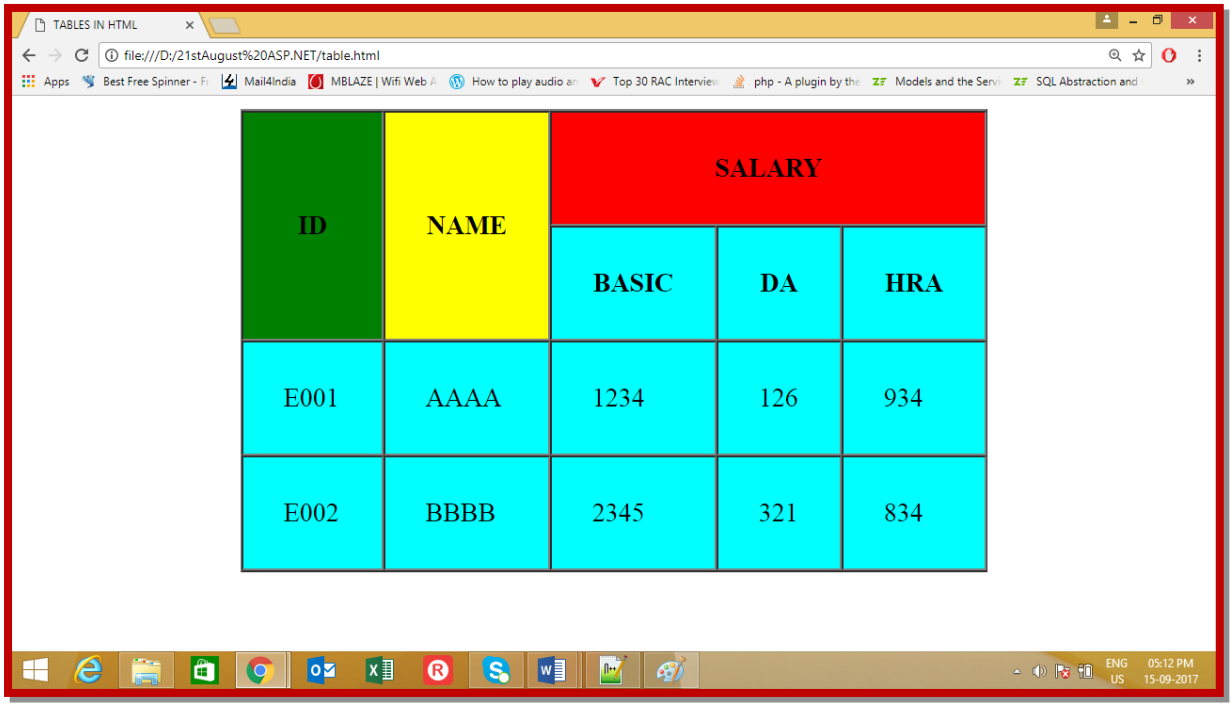
There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells. The cellspacing attribute defines space between table cells, while cellpadding represents the distance between cell borders and the content within a cell.

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.

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

Sample output:



HTML Forms

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.

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

The HTML **<form>** tag is used to create an HTML form

```
<form action = "Script URL" method = "GET|POST">
  form elements like input, textarea etc.
</form>
```

HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form −

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

The **action** attribute defines the action to be performed when the form is submitted.

Normally, the form data is sent to a web page on the server when the user clicks on the submit button.

The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data:

HTML Form Elements

The <input> Element

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

The <input> element can be displayed in several ways, depending on the **type** attribute.

For e.g.<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

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

The **<option>** elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the **selected** attribute to the option:

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

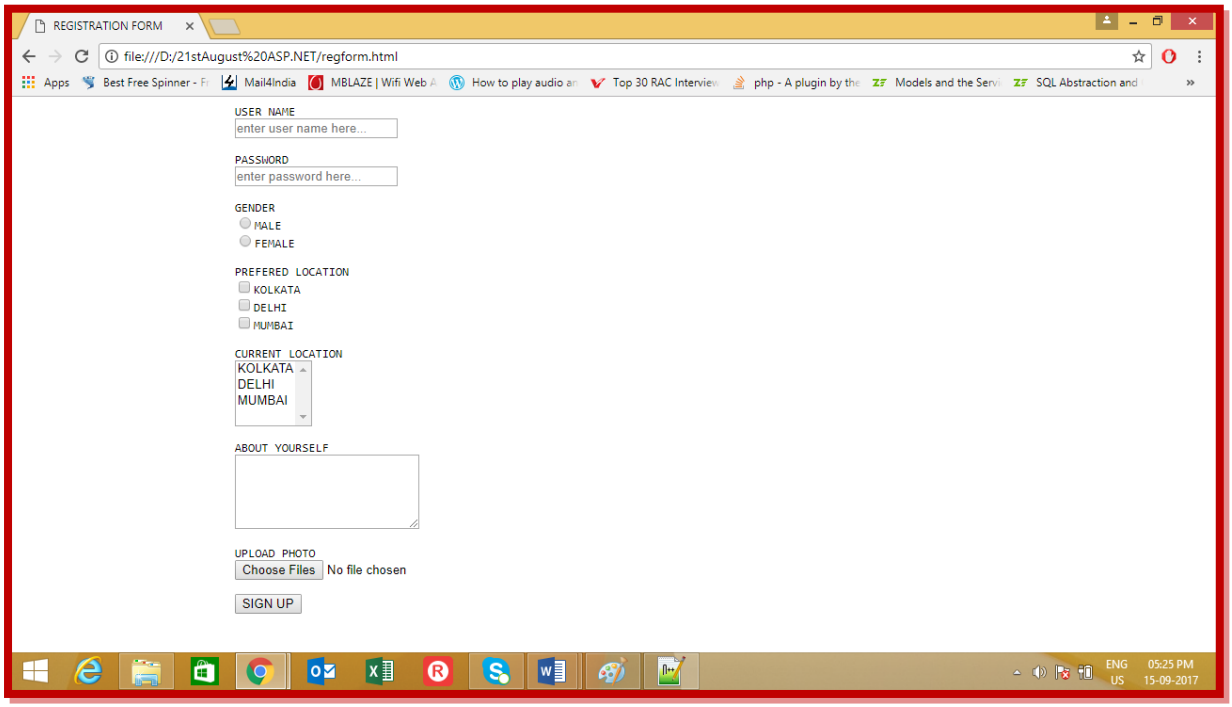
The **rows** attribute specifies the visible number of lines in a text area.

The **cols** attribute specifies the visible width of a text area.

```
<html>
	<head>
		<title>REGISTRATION FORM</title>
	</head>
	<body>
		<form method="post" action="#">
			<pre>
				<label>USER NAME</label>
				<input type="text" name="username" id="t1"
value=""
				placeholder="enter user name here...">

				<label>PASSWORD</label>
				<input type="password" name="pass" id="t2"
value=""
				placeholder="enter password here...">

				<label>GENDER</label>
				<input type="radio" name="gender" id="t3"
value="male">MALE
				<input type="radio" name="gender" id="t4"
value="female">FEMALE

				<label>PREFERED LOCATION</label>
				<input type="checkbox" name="c1" id="t5"
value="kol">KOLKATA
				<input type="checkbox" name="c2" id="t6"
value="del">DELHI
				<input type="checkbox" name="c3" id="t7"
value="mum">MUMBAI

				<label>CURRENT LOCATION</label>
				<select name="loc" id="t8" multiple>
```

```
                                <option value="kol">KOLKATA</option>
                                <option value="del">DELHI</option>
                                <option value="mum">MUMBAI</option>

                            </select>

                            <label>ABOUT YOURSELF</label>
                            <textarea rows="5" cols="25" name="ta" id="t9"
placeholder="enter text..">
                            </textarea>

                            <label>UPLOAD PHOTO</label>
                            <input type="file" name="f" id="t10" multiple>

                            <input type="submit" name="btn" id="t" value="SIGN
UP">
                        </pre>
                    </form>
                </body>
</html>
```

Output:



HTML5 Introduction

HTML5 is the most recent version of HTML. It is supported in all modern browsers.

What is New in HTML5?

The DOCTYPE declaration for HTML5 is very simple:

```
<!DOCTYPE html>
```

The character encoding (charset) declaration is also very simple:

```
<meta charset="UTF-8">
```

The default character encoding in HTML5 is UTF-8.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document.....
</body>

</html>
```

New Input Types

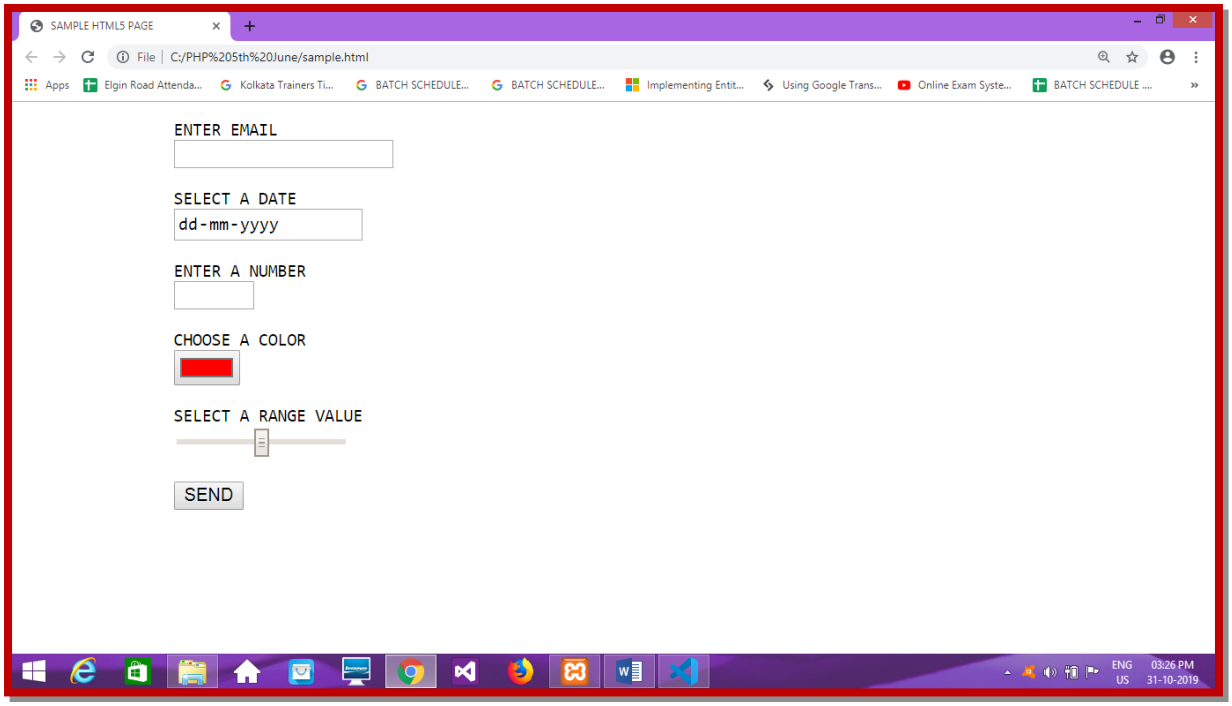
- color
- date
- datetime
- datetime-local
- email
- month

- number
- range
- search
- tel
- time
- url
- week

A sample program:

```
<!DOCTYPE html>
<html>
	<head>
		<title>SAMPLE HTML5  PAGE</title>
		<meta charset="utf-8">
	</head>
	<body>
		<form>
			<pre>
				<label>ENTER EMAIL ID</label>
				<input type="email" required>

				<label>SELECT A DATE</label>
				<input type="date" required>

				<label>ENTER A NUMBER</label>
				<input type="number" min="5" max="10" required>

				<label>SELECT A COLOUR</label>
				<input type="color">

				<label>SELECT A RANGE</label>
```

```
                        <input type="range" min="1000" max="10000">


                        <input type="submit" value="SAVE">
                    </pre>
                </form>
            </body>
</html>
```

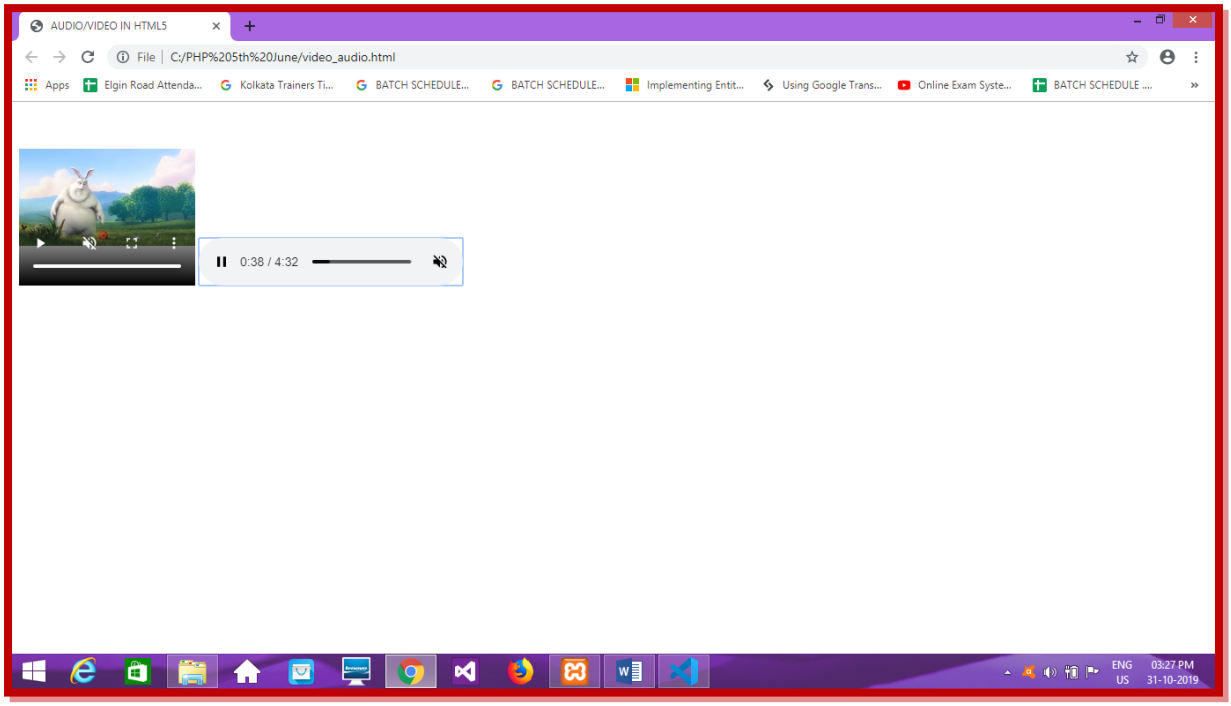


Audio/Video in HTML5

Sample program:

```
<!DOCTYPE html>
<html>
        <head>
                <title>AUDIO/VIDEO IN HTML5</title>
        </head>
        <body>
```

```
            <video controls width="300px" height="300px">
                <source src="images/a.mp4">
            </video>
            <audio controls autoplay>
                <source src="images/b.mp3">
            </audio>
        </body>
</html>
```



Using Style sheets: CSS

CSS is a language that describes the style of an HTML document.

CSS describes how HTML elements should be displayed.

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

Three Ways to Insert CSS

- External style sheet
- Internal style sheet
- Inline style

Inline Styles

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

```
<!DOCTYPE html>
<html>
	<head>
		<title>INLINE CSS</title>
		<meta charset="utf-8">
	</head>
	<body style="background-color:red;color:white">
		<h1 style="text-align:center">USING INLINE CSS</h1>
		<h2 style="color:yellow;text-align:center">SAMPLE
HEADING</h2>
		<p style="background-
color:green;width:400px;height:200px;margin-left:300px;
		border:5px solid white;line-height:150px;text-
align:center;font-weight:bold;


	font-style:italic;text-decoration:underline;border-top-left-
radius:50px;
		border-bottom-right-radius:50px">




			SOME SAMPLE TEXT IN A PARAGRAPH


		</p>


	</body>
</html>
```

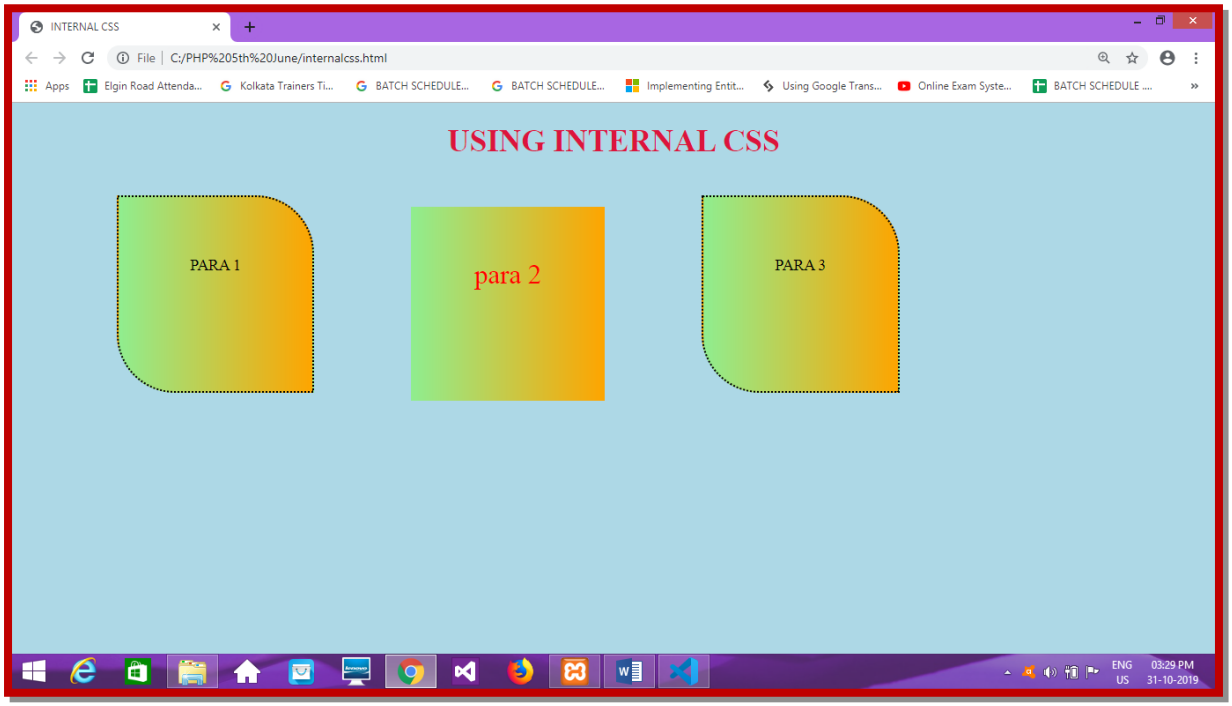
Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.

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

```
<!DOCTYPE html>
<html>
<head>
<title>INTERNAL CSS</title>
<style>
body{

background-color:lightblue;

}
h1
{
        color:crimson;
        text-align:center;
}
p
{
width:200px;
height:200px;
background:-webkit-linear-gradient(left,lightgreen,orange);
background:-moz-linear-gradient(left,lightgreen,orange);
background:-o-linear-gradient(left,lightgreen,orange);
background:-ms-linear-gradient(left,lightgreen,orange);
float:left;
margin-left:100px;
text-align:center;
line-height:140px;
}

.p1
{
        border-top-right-radius:60px;
        border-bottom-left-radius:60px;
        border:2px dotted black;
}
#p_id
{
        font-size:28px;
```

```
        color:red;
        text-transform:lowercase;

}
</style>
</head>
<body>
<h1>USING INTERNAL CSS</h1>
<p class="p1">
        PARA 1

</p>
<p id="p_id">
        PARA 2

</p>
<p class="p1">
        PARA 3

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



External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file!

Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

```
<!DOCTYPE html>
<html>
<head>
<title>EXTERNAL CSS</title>
<link href="style.css" rel="stylesheet">
</head>
<body>
<h1>USING EXTERNAL STYLESHEETS</h1>
<div id="wrapper">
        <div class="img" id="one">


        </div>
        <div class="img" id="two">


        </div>
        <p>
                Images may be subject to copyright &copy;
        </p>
</div>
</body>
</html>
```

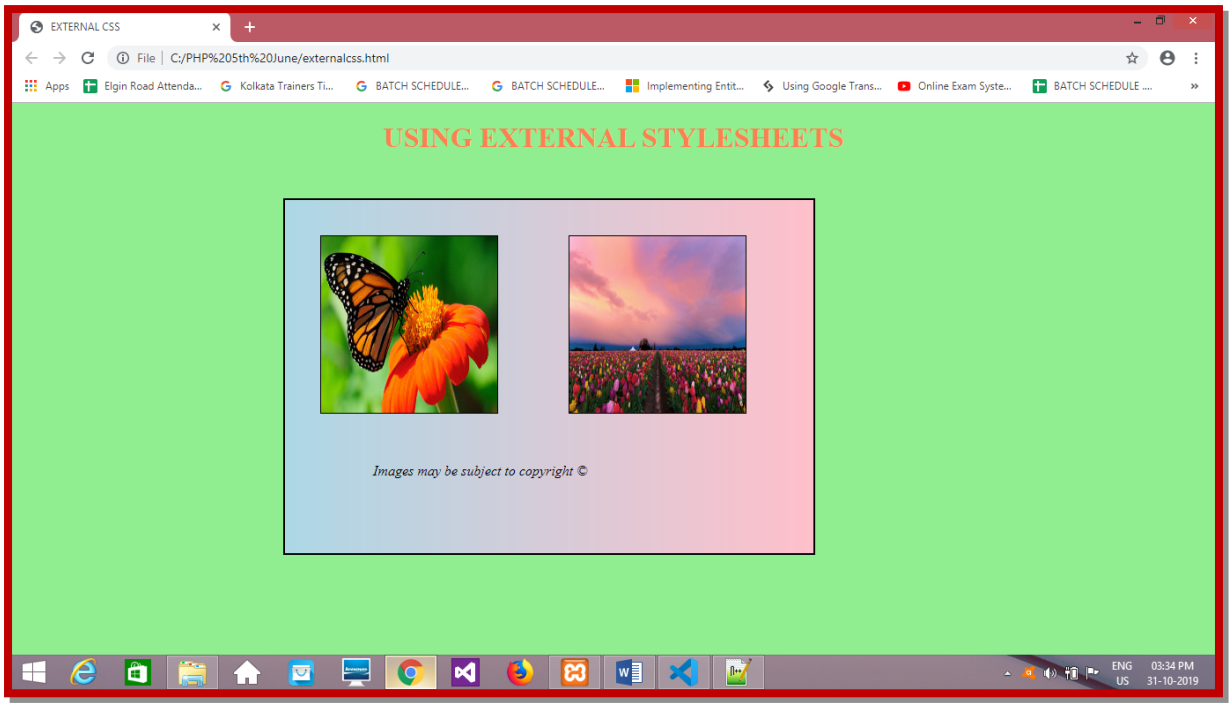
Style.css

```
body
{
        background-color:lightgreen;


}
```

```
h1
{
	text-align:center;
	color:coral;
}

#wrapper
{
	width:600px;
	height:400px;
	margin-left:300px;
	margin-top:50px;
	border:2px solid black;
	background:-webkit-linear-gradient(left,lightblue,pink);
}
.img
{
	width:200px;
	height:200px;
	float:left;
	margin:40px;
	background-size:100% 100%;
	border:1px solid black;
}
#one
{
	background-image:url('images/butterfly.jpg');
}
#two
{
	background-image:url('images/flowers.jpg');
}
p
{
```

```
        float:left;
        margin-left:100px;
        font-style:italic;
}
```

Sample output:



JavaScript is a lightweight programming language used to make a website interactive on the client side. It is very easy to implement because it is integrated with HTML.

In HTML, JavaScript programs are executed by the web browser.

JavaScript statements are separated by semicolons.

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **declare** variables.

JavaScript Comments

Code after double slashes **//** or between **/*** and ***/** is treated as a **comment**.

Comments are ignored, and will not be executed:

JavaScript is Case Sensitive

JavaScript Functions

A JavaScript function is a block of code used to perform a particular task.

JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

Creating pop up boxes in JavaScript

There are 3 types of pop up boxes in JavaScript

- Alert
- Prompt
- Confirm

Alert is used for displaying any output in a message box
Prompt is used to accept user input
Confirm is used to confirm the user action

In HTML, JavaScript code must be inserted between <script> and </script> tags.

Sample JavaScript program

```
<!DOCTYPE html>
```

```
<html>
   <head>
          <title>JAVASCRIPT BASIC FUNCTIONS</title>
          <meta charset="utf-8">
   </head>
   <body>
          <script>
                 alert("welcome to javascript");
                 var name=prompt("enter your name");
                 alert("Hello "+ name);
                 var result=confirm("Do you really wish to exit?");
                 if(result==true)
                        window.close();

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

Another example:

```
<!DOCTYPE html>
<html>
                <head>
                    <title>SUM OF 2 NUMBERS</title>
                    <meta charset="utf-8">
                </head>
                <body>
                    <script>
                           var a=parseInt(prompt("enter 1st
number"));
                           var b=parseInt(prompt("enter 2nd
number"));

                           var c=a+b;
                           alert("The sum is "+ c);

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

JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

Sample program

```
<!DOCTYPE html>
<html>
	<head>
		<title>SUM OF 2 NUMBERS USING HTML FORM</title>
		<meta charset="utf-8">
	</head>
	<body>
		<form>
			<pre>
				<label>ENTER 1ST NUMBER</label>
				<input type="text" name="t1" id="id1" value=""
				placeholder="enter 1st number here....">

				<label>ENTER 2ND NUMBER</label>
				<input type="text" name="t2" id="id2" value=""
				placeholder="enter 2nd number here....">

				<input type="submit" value="SUM" onclick="fun();">
			</pre>
		</form>
		<script>
			function fun()
			{
				var
a=parseInt(document.getElementById("id1").value);
				var
b=parseInt(document.getElementById("id2").value);
```

```
                alert("The sum is "+(a+b));
            }


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

In the above program we are passing form input to JavaScript and displaying the output in an alert box which is the sum of the two numbers input.

Using HTML, CSS and JavaScript

```
<!DOCTYPE html>
<html>
    <head>
        <title>SHOW/HIDE PARAGRAPH ON SINGLE BUTTON CLICK</title>
        <meta charset="utf-8">
        <style>
            #p1
            {
                width:200px;
                height:200px;
                background-color:red;
                color:white;
                border:5px double black;
                margin-left:400px;
                margin-top:100px;
                visibility:hidden;
            }
        </style>
    </head>
    <body>
```

```
		<p id="p1">
			SAMPLE PARAGRAPH
		</p>
		<form>
			<center>
				<input type="button" id="b1" value="SHOW"
onclick="fun1();">


			</center>
		</form>
		<script>
			function fun1()
			{
				if(document.getElementById("b1").value=="SHOW")
				{

	document.getElementById("p1").style.visibility="visible";
					document.getElementById("b1").value="HIDE";
				}
				else
if(document.getElementById("b1").value=="HIDE")
				{

	document.getElementById("p1").style.visibility="hidden";
					document.getElementById("b1").value="SHOW";
				}
			}
		</script>
	</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

External scripts are practical when the same code is reused in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

External scripts cannot contain <script> tags.

Example of form validation using external JavaScript

Validate.html

```
<html>
	<head>
		<title>FORMS IN HTML</title>
		<style>
			#error1,#error2{
				color:red;
				visibility:hidden;
			}

		</style>
		<script src="validate.js"></script>
	</head>
	<body>
		<form method="post" action="">
			<pre>
				<label>USER NAME</label>
				<input type="text" name="t1" id="id1" value=""
				placeholder="enter user name here...."
oninput="val_user();"><label id="error1">*User name required</label>

				<label>PASSWORD</label>
				<input type="password" name="t2" id="id2" value=""
				placeholder="enter password here...."
oninput="val_pass();"><label id="error2">*Password required</label>

				<input type="button" name="btn" id="id3"
value="LOGIN" onclick="validate();">

			</pre>
		</form>
```

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

validate.js

```
function validate()
{
	if(document.getElementById("id1").value=="")
		document.getElementById("error1").style.visibility="visible";

	if(document.getElementById("id2").value=="")
		document.getElementById("error2").style.visibility="visible";

}

function val_user()
{

	if(document.getElementById("id1").value=="")
		document.getElementById("error1").style.visibility="visible";
	else
		document.getElementById("error1").style.visibility="hidden";
}

function val_pass()
{

	if(document.getElementById("id2").value=="")
		document.getElementById("error2").style.visibility="visible";
	else
		document.getElementById("error2").style.visibility="hidden";
}
```

jQuery is a JavaScript Library that simplifies JavaScript programming.

It is easy to learn. It takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

Why jQuery?

There are lots of other JavaScript frameworks out there, but jQuery seems to be the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

How to Add JQuery to your pages:

- Download the jQuery library from jQuery.com

OR

- Include jQuery from a CDN, like Google

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed

- Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.4.1.min.js"></script>
</head>
```

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

Both Google and Microsoft host jQuery.

To use jQuery from Google or Microsoft, use one of the following:

Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
</head>
```

Microsoft CDN:

```
<head>
<script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.4.1.min.js"></script>
</head>
```

With jQuery you select (query) HTML elements and perform "actions" on them.

jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a document ready event:

```
$(document).ready(function(){

  // jQuery methods go here...

});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: $().

The element Selector

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

You can select all <p> elements on a page like this:

```
$("p")
```

The #id Selector

The jQuery *#id* selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

The .class Selector

The jQuery *.class* selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

What are Events?

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

The term **"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".

$(document).ready()

The $(document).ready() method allows us to execute a function when the document is fully loaded.

click()

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

dblclick()

The `dblclick()` method attaches an event handler function to an HTML element.

mouseenter()

The mouseenter() method attaches an event handler function to an HTML element.

mouseleave()

The mouseleave() method attaches an event handler function to an HTML element.

mousedown()

The mousedown() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element.

mouseup()

The mouseup() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

hover()

The hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

focus()

The focus() method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus

blur()

The blur() method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus

The on() Method

The on() method attaches one or more event handlers for the selected elements.

jQuery hide() and show()

With jQuery, you can hide and show HTML elements with the hide() and show() methods:

Syntax:

$(*selector*).hide(*speed,callback*);

$(*selector*).show(*speed,callback*);

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the hide() or show() method completes

jQuery toggle()

You can also toggle between hiding and showing an element with the toggle() method.

Shown elements are hidden and hidden elements are shown

Syntax:

$(*selector*).toggle(*speed,callback*);

The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after toggle() completes.

With jQuery you can fade elements in and out of visibility.

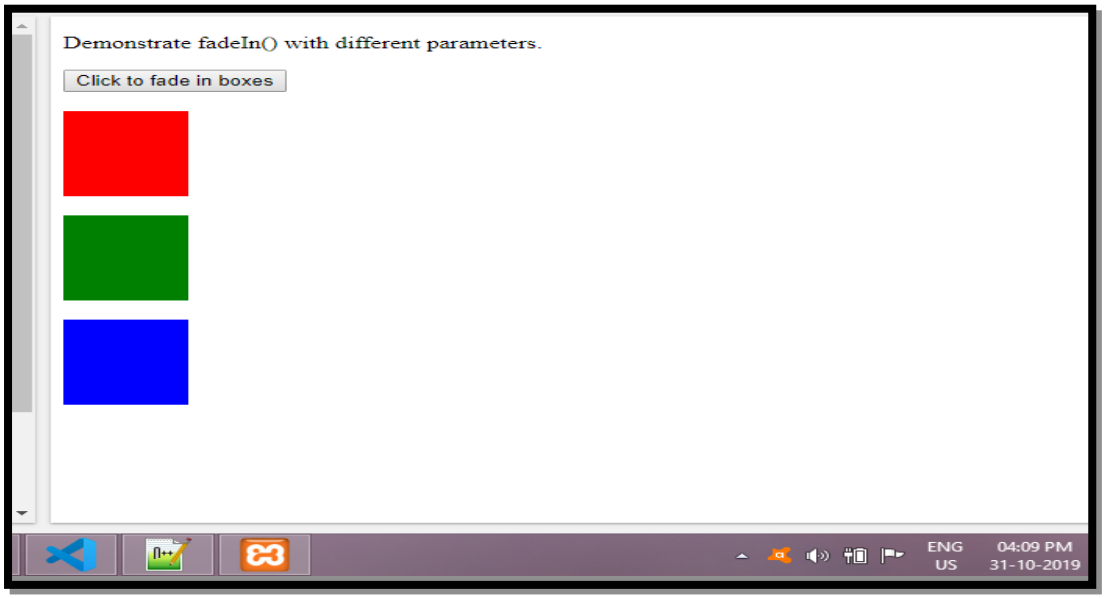
jQuery has the following fade methods:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

- The jQuery fadeIn() method is used to fade in a hidden element.

- **Syntax:**
- $(*selector*).fadeIn(*speed,callback*);
- The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
- The optional callback parameter is a function to be executed after the fading completes.

Sample output:



jQuery fadeOut() Method

The jQuery `fadeOut()` method is used to fade out a visible element.

Syntax:

$(*selector*).fadeOut(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

jQuery fadeToggle() Method

The jQuery `fadeToggle()` method toggles between the `fadeIn()` and `fadeOut()` methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

Syntax:

$(*selector*).fadeToggle(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

jQuery fadeTo() Method

The jQuery fadeTo() method allows fading to a given opacity (value between 0 and 1).

Syntax:

$(*selector*).fadeTo(*speed,opacity,callback*);

The required speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The required opacity parameter in the fadeTo() method specifies fading to a given opacity (value between 0 and 1).

The optional callback parameteris a function to be executed after the function completes.

The jQuery slide methods slide elements up and down

jQuery Sliding Methods

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

- slideDown()
- slideUp()
- slideToggle()

jQuery slideDown() Method

The jQuery slideDown() method is used to slide down an element.

Syntax:

$(*selector*).slideDown(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

jQuery slideUp() Method

The jQuery slideUp() method is used to slide up an element.

Syntax:

$(*selector*).slideUp(*speed,callback*);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

jQuery slideToggle() Method

The jQuery slideToggle() method toggles between the slideDown() and slideUp() methods.

If the elements have been slid down, slideToggle() will slide them up.

If the elements have been slid up, slideToggle() will slide them down.

$(*selector*).slideToggle(*speed,callback*);

The optional speed parameter can take the following values: "slow", "fast", milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

jQuery DOM Manipulation

One very important part of jQuery is the possibility to manipulate the DOM.

jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

DOM = Document Object Model

The DOM defines a standard for accessing HTML and XML documents:

Get Content - text(), html(), and val()

Three simple, but useful, jQuery methods for DOM manipulation are:

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

Set Content - text(), html(), and val()

We will use the same three methods from the previous page to **set content**:

- text() - Sets or returns the text content of selected elements
- html() - Sets or returns the content of selected elements (including HTML markup)
- val() - Sets or returns the value of form fields

Add New HTML Content

We will look at four jQuery methods that are used to add new content:

- append() - Inserts content at the end of the selected elements
- prepend() - Inserts content at the beginning of the selected elements
- after() - Inserts content after the selected elements
- before() - Inserts content before the selected elements

With jQuery, it is easy to remove existing HTML elements.

Remove Elements/Content

To remove elements and content, there are mainly two jQuery methods:

- remove() - Removes the selected element (and its child elements)
- empty() - Removes the child elements from the selected element

With jQuery, it is easy to manipulate the style of elements.

jQuery Manipulating CSS

jQuery has several methods for CSS manipulation. We will look at the following methods:

- addClass() - Adds one or more classes to the selected elements
- removeClass() - Removes one or more classes from the selected elements
- toggleClass() - Toggles between adding/removing classes from the selected elements
- css() - Sets or returns the style attribute

jQuery css() Method

The css() method sets or returns one or more style properties for the selected elements.

Set a CSS Property

To set a specified CSS property, use the following syntax:

css("*propertyname*","*value*");

Return a CSS Property

To return the value of a specified CSS property, use the following syntax:

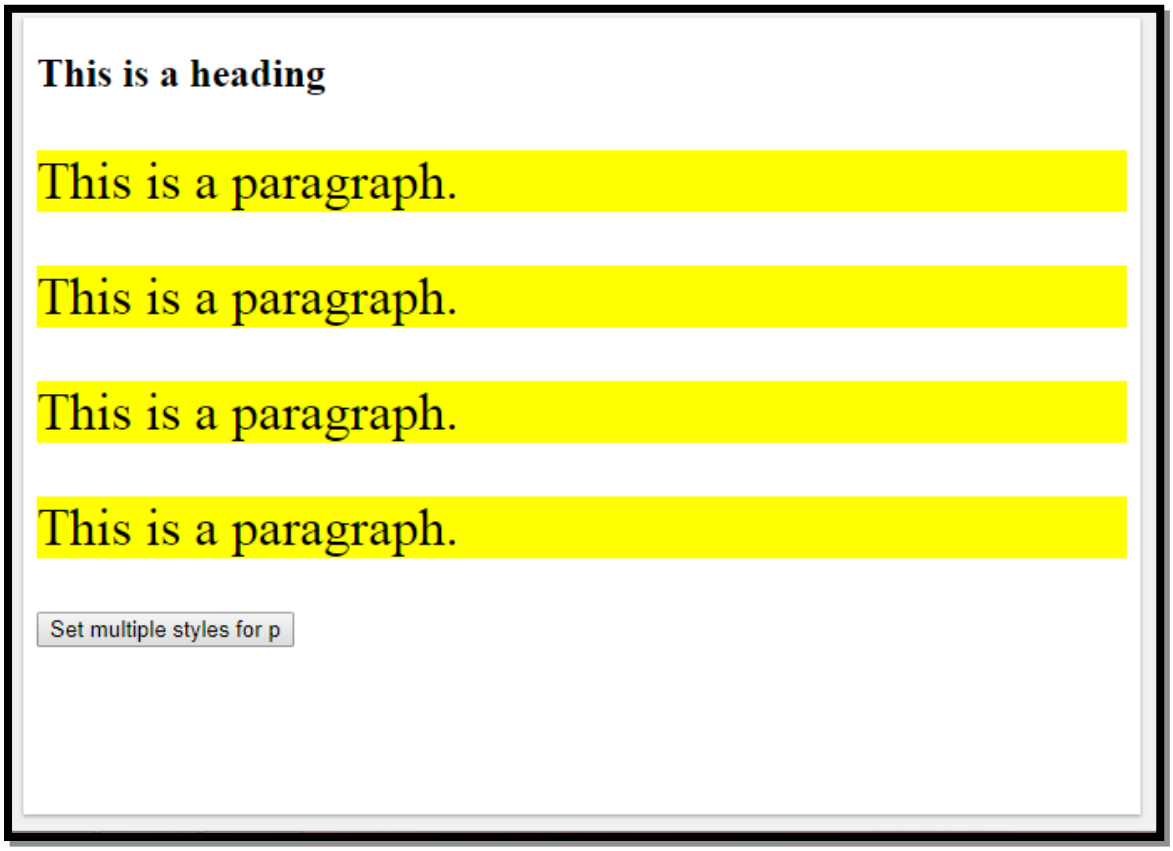
css("*propertyname*");

Set Multiple CSS Properties

To set multiple CSS properties, use the following syntax:

css({"*propertyname*":"*value*","*propertyname*":"*value*",...});

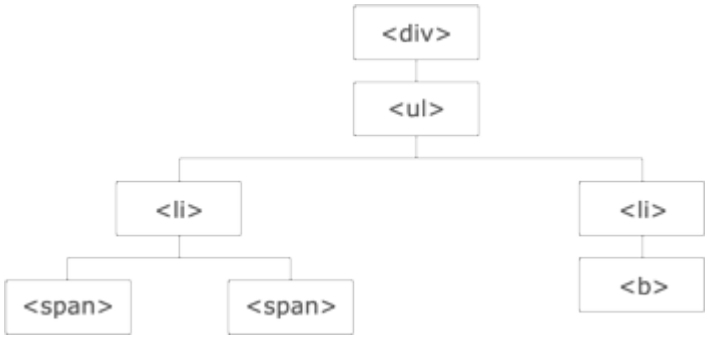
Sample output



What is Traversing?

jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.



- The element is the **parent** of <ul>, and an **ancestor** of everything inside of it
- The <ul> element is the **parent** of both <li> elements, and a **child** of
- The left <li> element is the **parent** of , **child** of <ul> and a **descendant** of
- The element is a **child** of the left <li> and a **descendant** of <ul> and
- The two <li> elements are **siblings** (they share the same parent)
- The right <li> element is the **parent** of <b>, **child** of <ul> and a **descendant** of
- The <b> element is a **child** of the right <li> and a **descendant** of <ul> and

Traversing the DOM

jQuery provides a variety of methods that allow us to traverse the DOM.

With jQuery you can traverse up the DOM tree to find ancestors of an element.

An ancestor is a parent, grandparent, great-grandparent, and so on.

Traversing Up the DOM Tree

Three useful jQuery methods for traversing up the DOM tree are:

- `parent()`
- `parents()`
- `parentsUntil()`

jQuery parent() Method

The `parent()` method returns the direct parent element of the selected element.

This method only traverse a single level up the DOM tree.

jQuery parents() Method

The parents() method returns all ancestor elements of the selected element, all the way up to the document's root element (<html>).

jQuery parentsUntil() Method

The parentsUntil() method returns all ancestor elements between two given arguments.

The following example returns all ancestor elements between a and a element:

With jQuery you can traverse down the DOM tree to find descendants of an element.

A descendant is a child, grandchild, great-grandchild, and so on.

Traversing Down the DOM Tree

Two useful jQuery methods for traversing down the DOM tree are:

- children()
- find()

jQuery children() Method

The children() method returns all direct children of the selected element.

This method only traverses a single level down the DOM tree.

jQuery find() Method

The find() method returns descendant elements of the selected element, all the way down to the last descendant.

With jQuery you can traverse sideways in the DOM tree to find siblings of an element.

Siblings share the same parent.

Traversing Sideways in The DOM Tree

There are many useful jQuery methods for traversing sideways in the DOM tree:

- siblings()
- next()
- nextAll()
- nextUntil()
- prev()
- prevAll()
- prevUntil()

jQuery siblings() Method

The siblings() method returns all sibling elements of the selected element.

jQuery next() Method

The next() method returns the next sibling element of the selected element.

jQuery nextAll() Method

The nextAll() method returns all next sibling elements of the selected element.

jQuery nextUntil() Method

The nextUntil() method returns all next sibling elements between two given arguments.

jQuery prev(), prevAll() & prevUntil() Methods

The prev(), prevAll() and prevUntil() methods work just like the methods above but with reverse functionality: they return previous sibling elements (traverse backwards along sibling elements in the DOM tree, instead of forward).


php

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.

- 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.

Common Uses of PHP

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. The other uses of PHP are:

- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.

- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

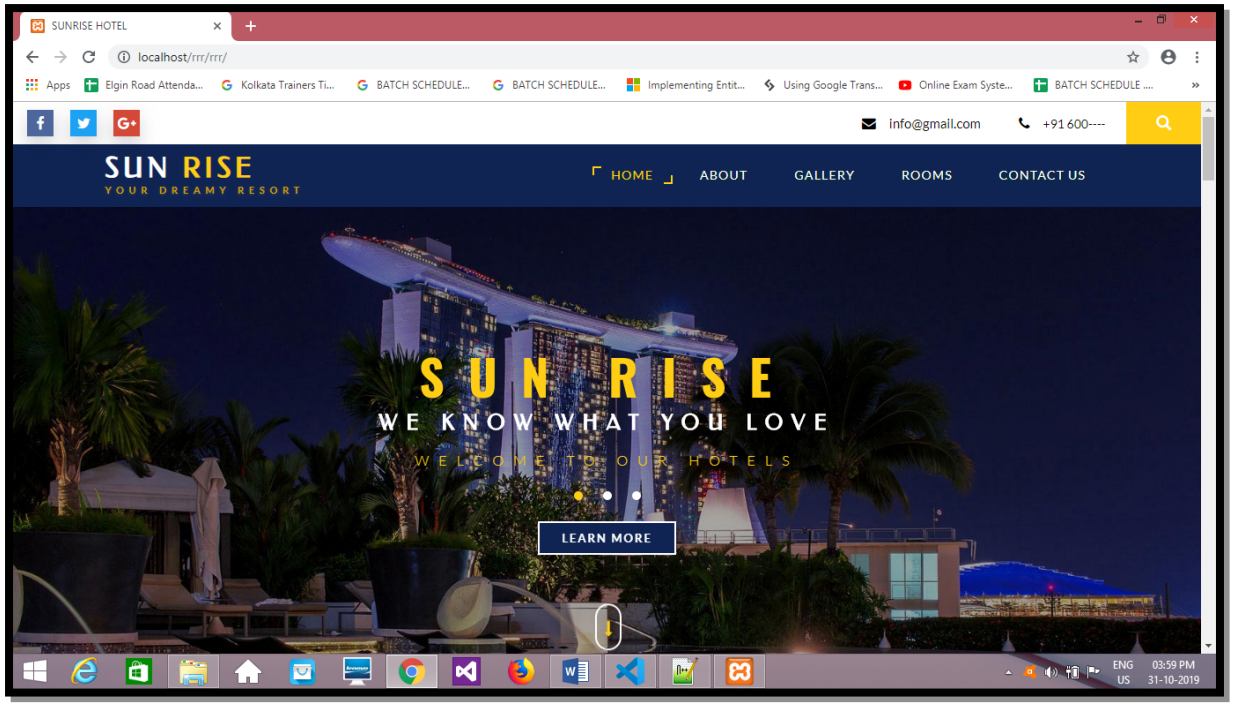
Characteristics of PHP

Five important characteristics make PHP's practical nature possible:

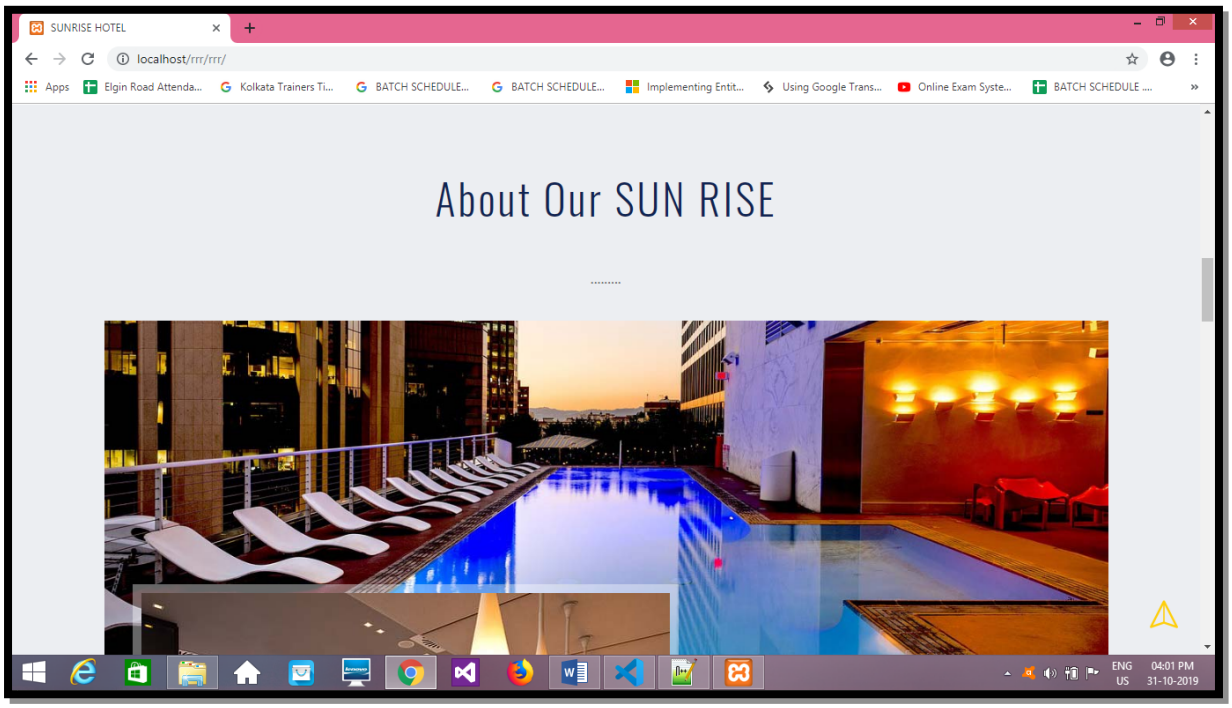
- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

A sample application on Hotel Booking using PHP &MySQL

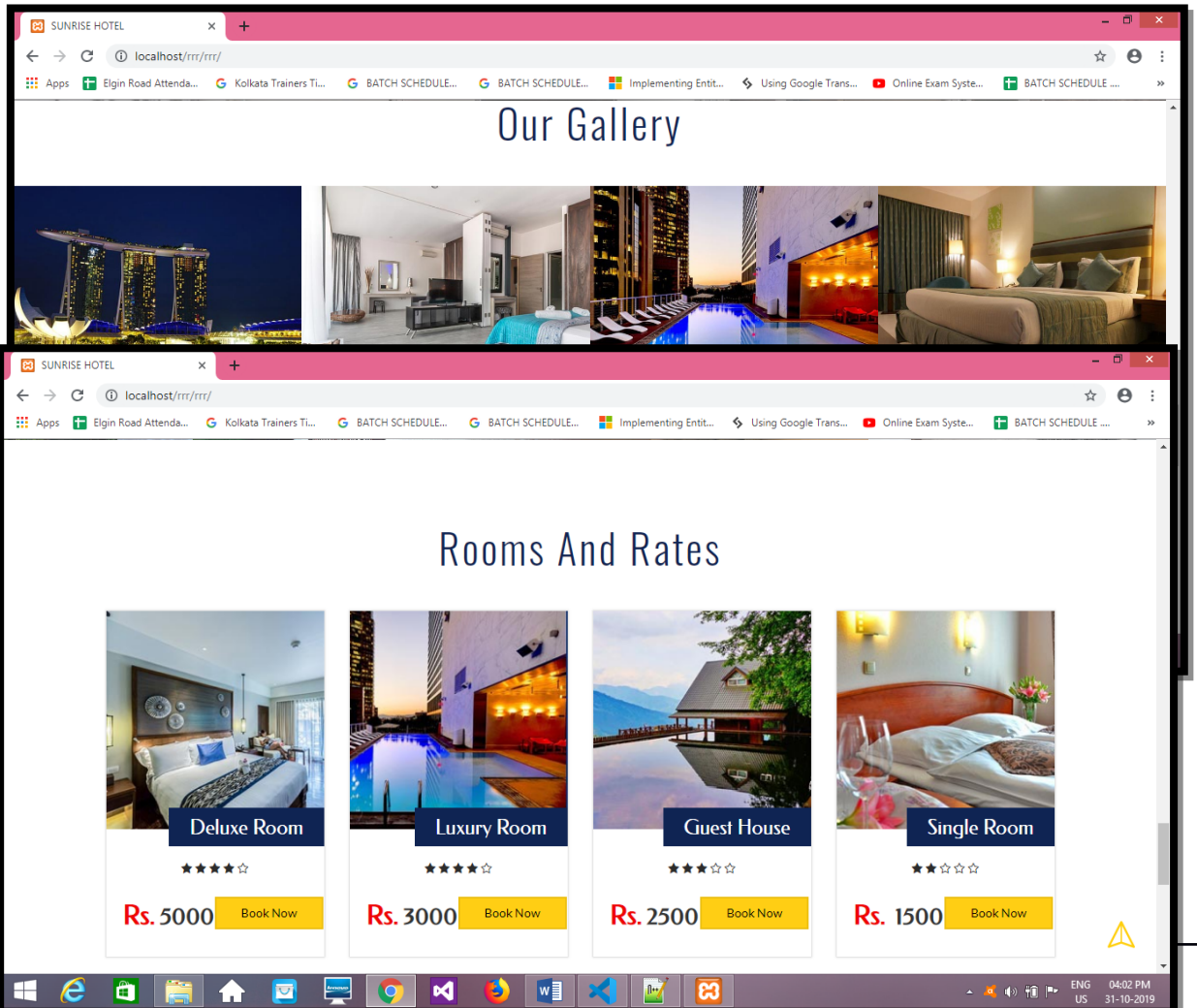
Homepage



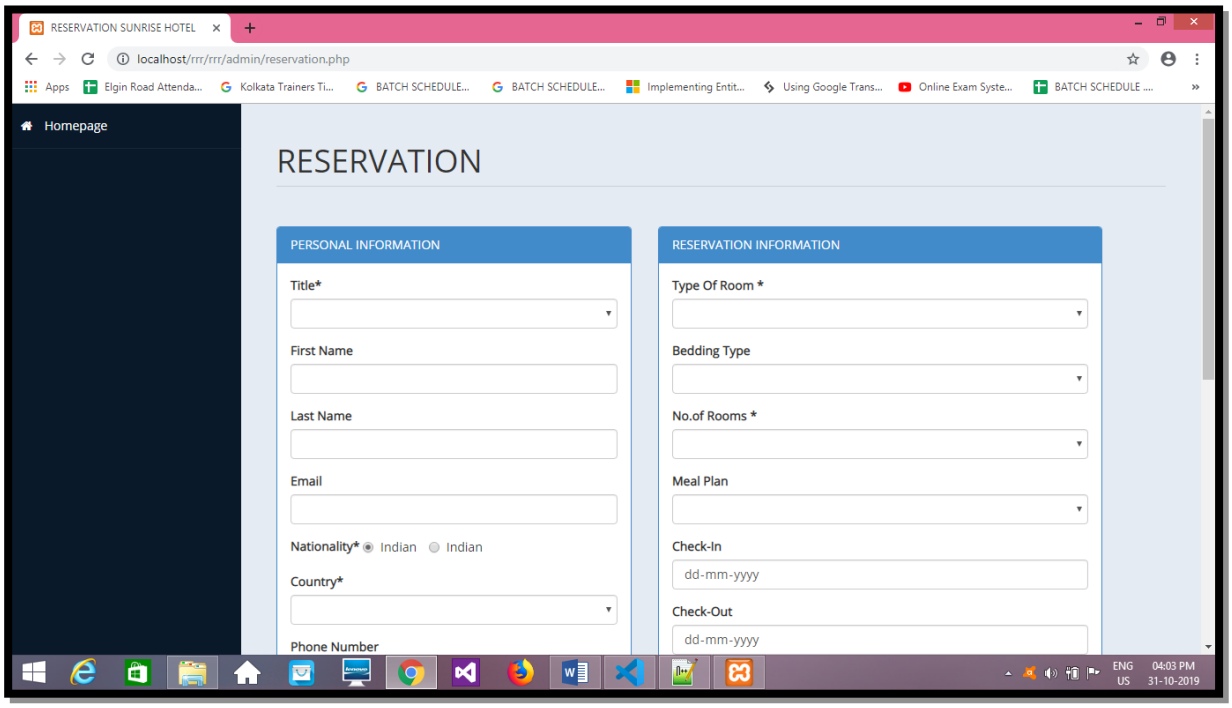
About



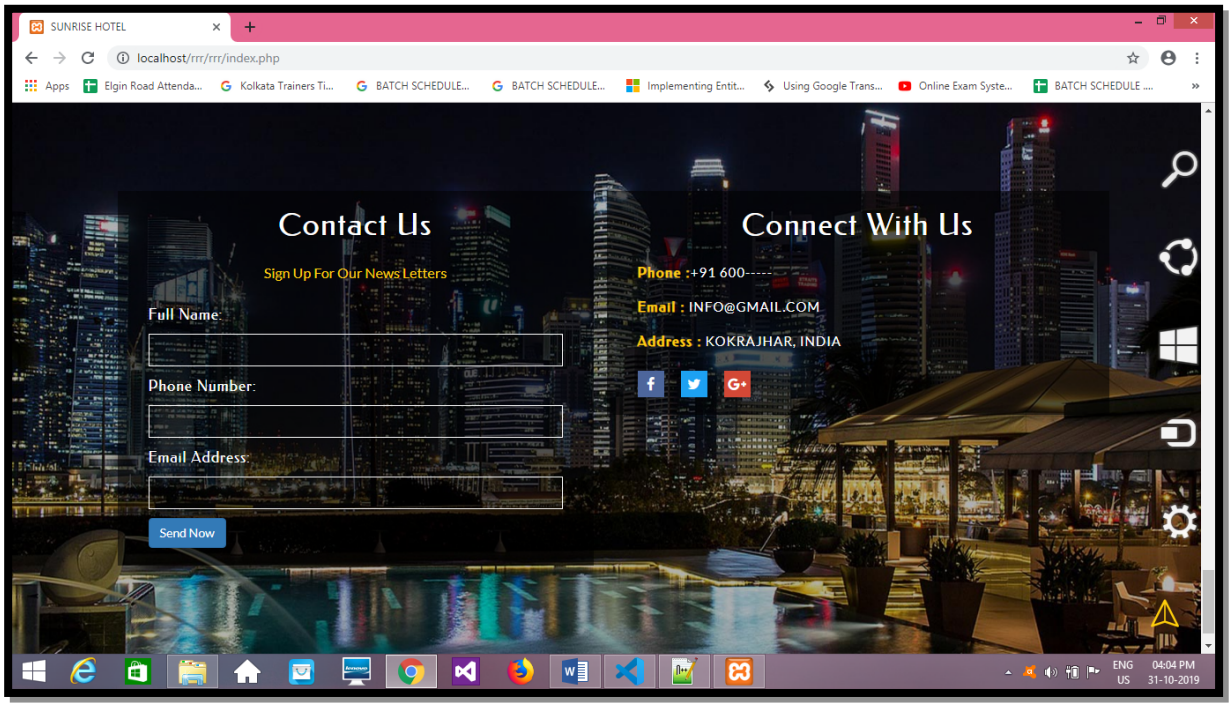
Gallery



Reservation



Contact



Start Learning now….

"Hello World" Script in PHP

To get a feel of PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal

HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>

<head>
```

```
<title>Hello World</title>
<body>
    <?php echo "Hello, World!";?>
</body>
</html>
```

It will produce the following result:

Hello, World!

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ate are recognized by the PHP Parser.

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

<?    PHP code goes here ?>
```

Most common tag is the <?php...?> and we will also use the same tag in our tutorial.

From the next chapter, we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

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

1. Text Editor like Notepad++

2. XAMPP Server
3. Web Browser preferably Google Chrome, Mozilla Firefox

Step 1:

Write your program in any text editor like Notepad++ and save it in the location:

C:/xampp/htdocs/your folder

Step 2:

Start Apache & MySQL modules in XAMPP Control Panel

Step 3:

Open any browser and type the following url:

localhost/your folder/filename.php

Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' There are four ways to do this:

Canonical PHP tags

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags

Short or short-open tags look like this:

<?...?>

Short tags are, as one might expect, the shortest **option You must** do one of two things to enable PHP to recognize the tags:

- Choose the --enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

Commenting PHP Code

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevantto the local code. Here are the examples of single line comments.

```
<?

#This is a comment, and

#This is the second line of the comment

/   This is a comment too. Each style
comments only print "An example with single
line comments";

?>
```

Multi-lines printing: Here are the examples to print multiple lines in a single printstatement:

```
<?

#First
Example
print
<<<END

This uses the "here document" syntax to
output multiple lines with $variable
interpolation. Note that the here document
terminator must appear on a line with just a
semicolon no extra whitespace! END;

#Second Example

print "This spans

multiple lines. The newlines will be


output as well";

?>
```

Multi-lines comments: They are generally used to provide pseudocode algorithms andmore detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?

/* This is a comment with multiline

    Author : Mohammad Mohtashim

    Purpose: Multiline Comments Demo


    Subject: PHP

*/

print "An example with multi line
comments"; ?>
```

PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row.one whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent:

```
$fou = 2 +
r    2;        // single spaces

$fou
r    <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs

$fou
r    =

2+

     multipl
2; // e        lines
```

PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out the following example:

```
<html>

<body>

<?

$capital = 67;

print("Variable capital is $capital<br>");

print("Variable CaPiTaL is $CaPiTaL<br>");

?>

</body>

</html>
```

This will produce the following result:

Variable capital is 67

Variable CaPiTaL is

Statements are expressions terminated by semicolons

A *statement* in PHP is any expression that is followed by a semicolon (;).Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting:

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces. Here both statements are equivalent:

```
if (3 == 2 + 1)

   print("Good - I haven't totally lost my mind.<br>");


if (3 == 2 + 1)

{

    print("Good - I haven't totally");

    print("lost my mind.<br>");

}
```

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.

- PHP does a good job of automatically converting types from one to another when necessary.

- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.

- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.

- **Booleans:** have only two possible values either true or false.

- **NULL:** is a special type that only has one value: NULL.

- **Strings:** are sequences of characters, like 'PHP supports string operations.'

- **Arrays:** are named and indexed collections of other values.

- **Objects:** are instances of programmer-defined classes, which can package up bothother kinds of values and functions that are specific to the class.

- **Resources:** are special variables that hold references to resources external to PHP(such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simile data type in this chapters. Array and Objects will be explained separately.

Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
$int_var = 12345;

$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is . (2**31 . 1) (or .2,147,483,647).

Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
$many = 2.2888800;

$many_2 = 2.2111200;

$few = $many + $many_2;

print(.$many + $many_2 = $few<br>.);
```

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
if (TRUE)
    print("This will always print<br>");
else
    print("This will never print<br>");
```

Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.

- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;

$true_str = "Tried and true"

$true_array[49] = "An array element";

$false_array = array();

$false_null = NULL;

$false_num = 999 - 999;

$false_str = "";
```

NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string:

```
$string_1 = "This is a string in double quotes";

$string_2 = "This is a somewhat longer, singly quoted
string"; $string_39 = "This string has thirty-nine
characters"; $string_0 = ""; // a string with zero
characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?

$variable = "name";

$literally = 'My $variable will not print!\\n';
print($literally);

$literally = "My $variable will print!\\n";

print($literally);

?>
```

This will produce the following result:

```
My $variable will not print!\n

My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Variable Naming

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc

There is no size limit for variables.

PHP – Variables

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

- Local variables
- Function parameters
- Global variables

- Static variables

PHP Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?

$x = 4;

function assignx () {

$x = 0;

print "\$x inside function is $x.
";

}

assignx();

print "\$x outside of function is $x.

";

?>
```

This will produce the following result.

```
$x inside function is 0.

$x outside of function is 4.
```

PHP Function Parameters

PHP Functions are covered in detail in PHP Function Chapter. In short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return a value.

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?

/   multiply a value by 10 and return it to the
caller function multiply ($value) {

    $value = $value *
    10; return $value;

}



$retval = multiply (10);

Print "Return value is $retval\n";

?>
```

This will produce the following result.

```
Return value is 100
```

PHP Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as

global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example:

```
<?

$somevar = 15;

function addit() {

GLOBAL $somevar;

$somevar++;

print "Somevar is $somevar";

}

addit();

?>
```

This will produce the following result.

```
Somevar is 16
```

PHP Static Variables

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?

function keep_track() {

    STATIC $count = 0;

    $count++;

    print $count;

    print "

";

}

keep_track();

keep_track();

keep_track();
```

```
?>
```

This will produce the following result.

```
1

2

3
```

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e., it is stored in a variable or returned by a function.

constant() example

```
<?php

define("MINSIZE", 50);

echo MINSIZE;

echo constant("MINSIZE"); // same thing as the previous line

?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are

- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.

- Constants may be defined and accessed anywhere without regard to variable scoping rules.

- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names

```
// Valid constant names

define("ONE",     "first thing");

define("TWO2", "second thing");

define("THREE_3", "third thing")

// Invalid constant names

define("2TWO", "second thing");

define("__THREE__", "third value");
```

PHP Magic constants

PHP provides a large number of predefined constants to any script which it runs.

There are five magical constants that change depending on where they are used. For example, the value of __LINE__ depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

The following table lists a few "magical" PHP constants along with their description:

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP

	4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances.
__FUNCTION__	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

What is Operator? Simple answer can be given using expression*4 + 5 is equal to 9*.Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

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

Operators Let’s have a look on

all operators one by one.

Arithmetic Operators

The following arithmetic operators are supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide the numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by	A++ will give 11

	one	
--	Decrement operator, decreases integer value by one	A-- will give 9

Example

Try the following example to understand all the arithmetic operators. Copy and paste following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title>Arithmetical Operators</title><head>

<body>

<?php

    $a = 42;

    $b = 20;



    $c = $a + $b;

    echo "Addition Operation Result: $c
    <br/>"; $c = $a - $b;

    echo "Subtraction Operation Result: $c
    <br/>"; $c = $a * $b;

    echo "Multiplication Operation Result: $c
    <br/>"; $c = $a / $b;

    echo "Division Operation Result: $c
    <br/>"; $c = $a % $b;

    echo "Modulus Operation Result: $c
    <br/>"; $c = $a++;

    echo "Increment Operation Result: $c
    <br/>"; $c = $a--;

    echo "Decrement Operation Result: $c <br/>";
?></body>

</html>
```

This will produce the following result:

```
Addition Operation Result: 62

Subtraction Operation Result: 22

Multiplication Operation Result: 840

Division Operation Result: 2.1

Modulus Operation Result: 2

Increment Operation Result: 42

Decrement Operation Result: 43
```

Comparison Operators

There are following comparison operators supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right	(A >= B) is not true.

	operand, if yes then condition becomes true.	
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.	(A <= B) is true.

Example

Try the following example to understand all the comparison operators. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title>Comparison Operators</title><head>

<body>

<?php

    $a = 42;

    $b = 20;


    if( $a == $b ){

        echo "TEST1 : a is equal to b<br/>";

    }else{

        echo "TEST1 : a is not equal to b<br/>";
```

```
}


if( $a > $b ){

   echo "TEST2 : a is greater than
b<br/>"; }else{

   echo "TEST2 : a is not greater than b<br/>";

}

if( $a < $b ){

   echo "TEST3 : a is less than b<br/>";

}else{

   echo "TEST3 : a is not less than b<br/>";

}

if( $a != $b ){

   echo "TEST4 : a is not equal to
b<br/>"; }else{

   echo "TEST4 : a is equal to b<br/>";

}

if( $a >= $b ){

   echo "TEST5 : a is either greater than or equal to
b<br/>"; }else{

   echo "TEST5 : a is neither greater than nor equal to b<br/>";
```

```
    }

    if( $a <= $b ){

        echo "TEST6 : a is either less than or equal to
    b<br/>"; }else{

        echo "TEST6 : a is neither less than nor equal to b<br/>";

    }

?>

</body>

</html>
```

This will produce the following result:

```
TEST1 : a is not equal to b

TEST2 : a is greater than b

TEST3 : a is not less than b

TEST4 : a is not equal to b

TEST5 : a is either greater than or equal to b
```

TEST6 : a is neither less than nor equal to b

Logical Operators

The following logical operators are supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true, then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero, then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A || B) is true.

!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is false.

Example

Try the following example to understand all the logical operators. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title>Logical Operators</title><head>

<body>

<?php

    $a = 42;
```

```
$b =
0;


if( $a && $b ){

   ech "TEST1 : Both a and b are
   o    true<br/>";

}else{

   ech "TEST1 : Either a or b is
   o    false<br/>";

}

if( $a and $b ){

   ech "TEST2 : Both a and b are
   o    true<br/>";

}else{

   ech "TEST2 : Either a or b is
   o    false<br/>";

}

if( $a || $b ){

   ech "TEST3 : Either a or b is
   o    true<br/>";

}else{

   ech "TEST3 : Both a and b are
   o    false<br/>";

}

if( $a or $b ){

   ech "TEST4 : Either a or b is
   o    true<br/>";

}else{

   ech "TEST4 : Both a and b are
   o    false<br/>";

}

$a =
10;

$b =
20;

if( $a ){
```

```
    ech
    o     "TEST5 : a is true <br/>";

}else{

    ech  "TEST5 :
    o     a             is false<br/>";

}

if( $b ){

    ech
    o     "TEST6 : b is true <br/>";

}else{
```

```
    ech  "TEST6 :
    o     b             is false<br/>";

}

if( !$a  ){

    ech
    o     "TEST7 : a is true <br/>";

}else{

    ech  "TEST7 :
    o     a             is false<br/>";
```

```
    }
    if( !$b ){
        echo "TEST8 : b is true <br/>";
    }else{
        echo "TEST8 : b is false<br/>";
    }
?>
</body>
</html>
```

This will produce the following result:

```
TEST1 : Either a or b is false
TEST2 : Either a or b is false
TEST3 : Either a or b is true
TEST4 : Either a or b is true
TEST5 : a is true
TEST6 : b is true
TEST7 : a is false
TEST8 : b is false
```

Assignment Operators

PHP supports the following assignment operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A

*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Example

Try the following example to understand all the assignment operators. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title>Assignment Operators</title><head>

<body>

<?php

    $a = 42;

    $b = 20;
```

```
$        $a +            /* Assignment
c  =      $b;                  operator  */

ech   "Addition Operation Result:
o                                  $c<br/>";

$  +=            c value was 42 +
c  $a;      /*                  20 = 62  */

echo       AN     Assignment Operation Result: $c
"Add         D                                    <br/>";


$c -= $a; /* c value was 42 + 20 + 42 = 104 */


echo "Subtract AND Assignment Operation Result:
$c <br/>"; $c *= $a; /* c value was 104 - 42 = 62 */


echo "Multiply AND Assignment Operation Result: $c
<br/>"; $c /= $a; /* c value was 62 * 42 = 2604 */


echo "Division AND Assignment Operation Result: $c
<br/>"; $c %= $a; /* c value was 2604/42 = 62*/


echo "Modulus AND Assignment Operation Result: $c <br/>";

?></body>

</html>
```

This will produce the following result:

```
Addition Operation Result: 62

Add AND Assignment Operation Result: 104

Subtract AND Assignment Operation Result: 62

Multiply AND Assignment Operation Result: 2604

Division AND Assignment Operation Result: 62

Modulus AND Assignment Operation Result: 20
```

Conditional Operator

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

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Try the following example to understand the conditional operator. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>

<head><title> Operators</title><head>
```

```
<body>

<?php

    $a = 10;

    $b = 20;



    /* If condition is true then assign a to result otherwise b
    */ $result = ($a > $b ) ? $a :$b;

    echo "TEST1 : Value of result is $result<br/>";

    /* If condition is true then assign a to result otherwise b
    */ $result = ($a < $b ) ? $a :$b;

    echo "TEST2 : Value of result is $result<br/>";

?>

</body>

</html>
```

This will produce the following result:

```
TEST1 : Value of result is 20

TEST2 : Value of result is 10
```

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports the following three decision making statements:

- **if...else statement** - use this statement if you want to execute a set of code whena condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true
- **switch statement** - is used if you want to select one of many blocks of code to beexecuted, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

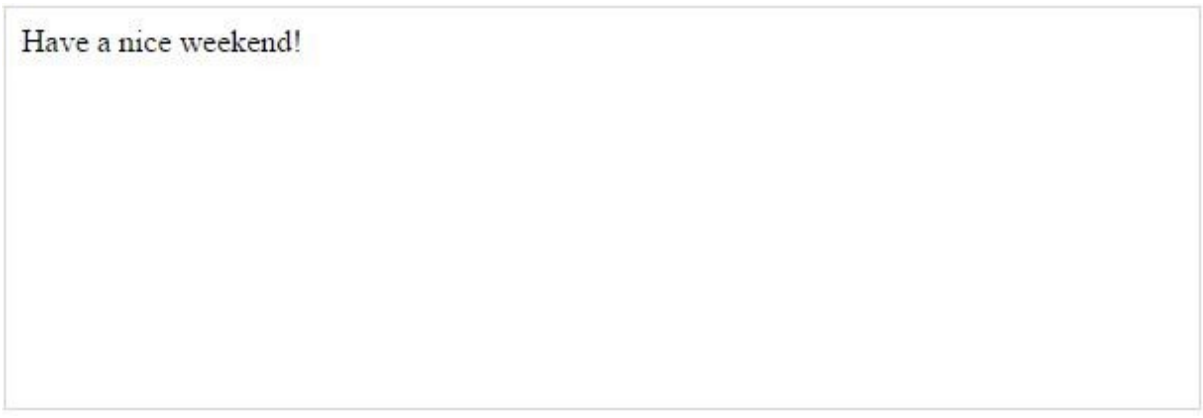
```
if (condition)
   code to be executed if condition is true;
else
   code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>

<body>

    <?php

$d=date("D");

if ($d=="Fri")

  echo "Have a nice weekend!";

else

  echo "Have a nice day!";

?>

    </body>


</html>
```

It will produce the following result:



The ElseIf Statement

If you want to execute some code if one of the several conditions is true, then use the elseif statement.

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;
```

Example

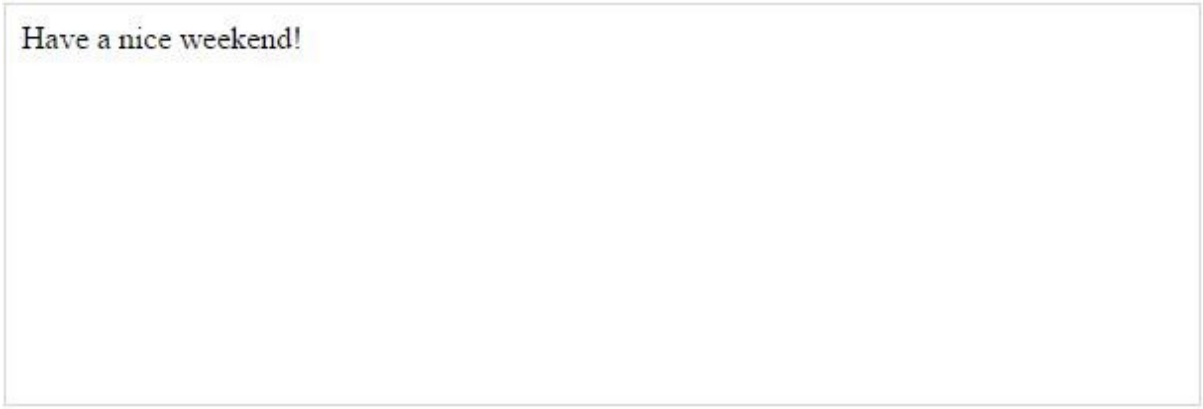
The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>
    <?php
$d=date("D");
if ($d=="Fri")
```

```
    echo "Have a nice weekend!";
```

```
elseif ($d=="Sun")

   echo "Have a nice Sunday!";

else

   echo "Have a nice day!";

?>

      </body>

</html>
```

It will produce the following result:



The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression)

{

case label1:

  code to be executed if expression =
  label1; break;

case label2:

  code to be executed if expression =
  label2; break;

default:

  code to be executed

  if expression is different

  from both label1 and label2;

}
```

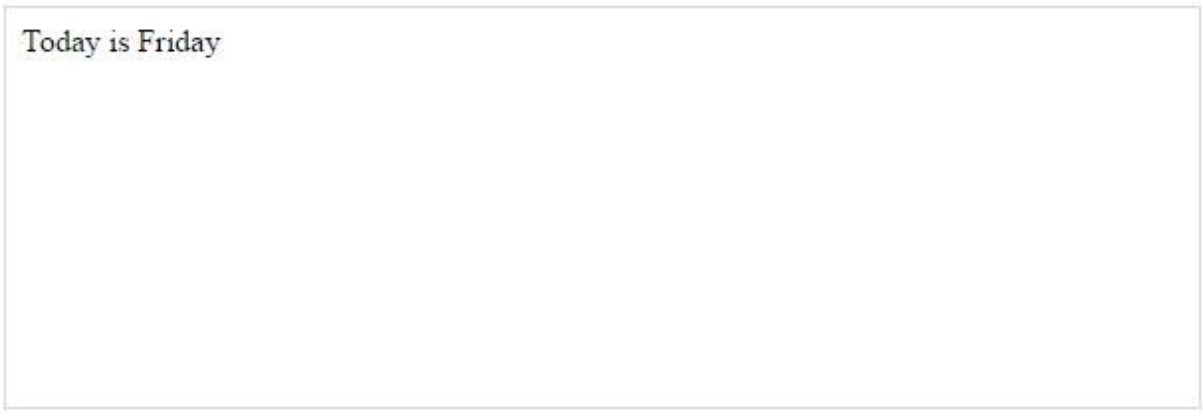
Example

The *switch* statement works in an unusual way. First it evaluates the given expression, then seeks a label to match the resulting value. If a matching value is found, then the code associated with the matching label will be executed. If none of the labels match, then the statement will execute any specified default code.

```
<html>

<body>

    <?php

$d=date("D");

switch ($d)

{

case "Mon":

  echo "Today is Monday";

  break;

case "Tue":

  echo "Today is Tuesday";

  break;

case "Wed":

  echo "Today is Wednesday";

  break;

case "Thu":

  echo "Today is Thursday";
```

```
   break;
  case "Fri":
   echo "Today is Friday";
   break;
  case "Sat":
   echo "Today is Saturday";
   break;
  case "Sun":
   echo "Today is Sunday";
   break;
  default:
   echo "Wonder which day is this ?";
  }
  ?>
      </body>
  </html>
```

It will produce the following result:



Loops

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for -** loops through a block of code a specified number of times.
- **while -** loops through a block of code if and as long as a specified condition is true.
- **do...while -** loops through a block of code once, and then repeats the loop as longas a special condition is true.
- **foreach -** loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations.

A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

```
<html>

<body>

<?php

$a = 0;


$b = 0;



for( $i=0; $i<5; $i++ )

{

    $a += 10;


    $b += 5;

}

echo ("At the end of the loop a=$a and b=$b" );

?>

</body>

</html>
```

This will produce the following result:

At the end of the loop a=50 and b=25

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true, then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

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

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation becomes false and the loop ends.

```
<html>
<body>
<?php
$i = 0;
$num = 50;
```

```
while( $i < 10)

{

    $num--;

    $i++;

}
```

```
echo ("Loop stopped at i = $i and num = $num" );


?>

</body>

</html>
```

This will produce the following result:

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it will then repeat the loop as long as a condition is true.

Syntax

```
do

{

    code to be executed;

}while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable **i** as long as it has a value of less than 10:

```
<html>

<body>

<?php

$i = 0;

$num = 0;

do

{

  $i++;

}while( $i < 10 );

echo ("Loop stopped at i = $i" );

?>

</body>

</html>
```

This will produce the following result:

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value)

{

    code to be executed;



}
```

Example

Try out the following example to list out the values of an array.

```
<html>

<body>

<?php

$array = array( 1, 2, 3, 4, 5);

foreach( $array as $value )

{

   echo "Value is $value <br />";
```

```
}

?>

</body>

</html>
```

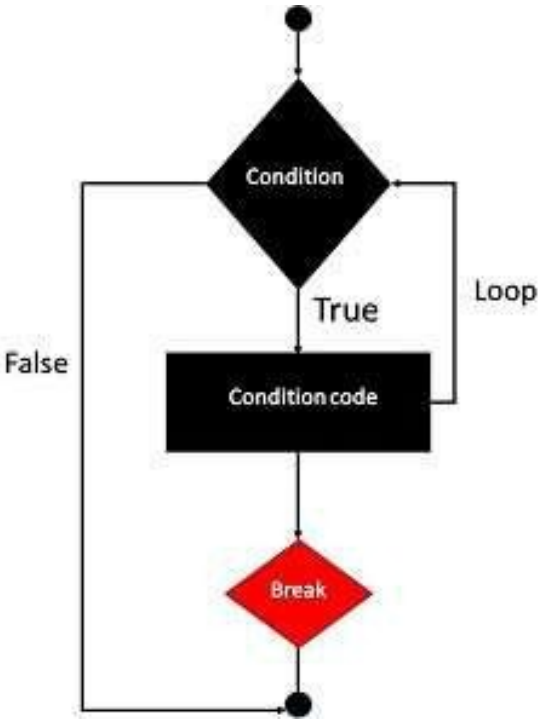
This will produce the following result:

```
Value is 1

Value is 2

Value is 3

Value is 4

Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example, the condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>

<body>
```

```
<?php

$i = 0;


while( $i < 10)

{

    $i++;

    if( $i == 3 )break;

}

echo ("Loop stopped at i = $i" );

?>

</body>

</html>
```

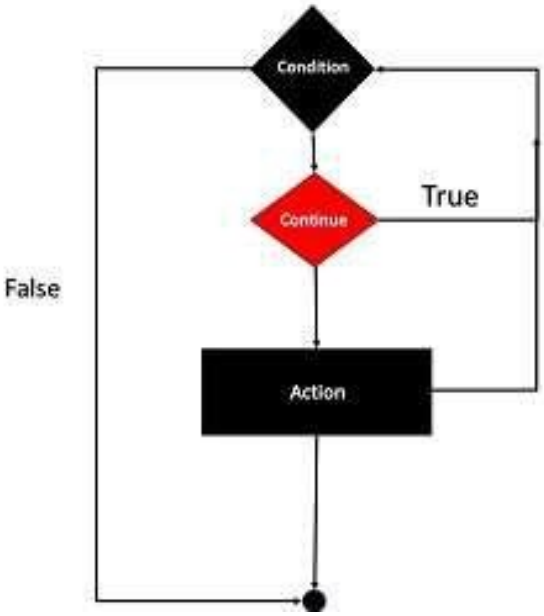
This will produce the following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example, the loop prints the value of array, but when the condition becomes true, it just skips the code and next value is printed.

```
<html>

<body>

<?php

$array = array( 1, 2, 3, 4, 5);

foreach( $array as $value )

{

   if( $value == 3 )continue;

   echo "Value is $value <br />";

}

?>

</body>

</html>
```

This will produce the following result:

```
Value is 1

Value is 2

Value is 4

Value is 5
```

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessedin linear fashion
- **Associative array** - An array with strings as index. This stores element values inassociation with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values areaccessed using multiple indices

NOTE: Built-in array functions is given in function referencePHP Array Functions

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, the array index starts from zero.

Example

The following example demonstrates how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>

<body>

<?php

/* First method to create array. */

$numbers = array( 1, 2, 3, 4, 5);

foreach( $numbers as $value )

{

   echo "Value is $value <br />";

}

/* Second method to create array. */

$numbers[0] = "one";

$numbers[1] = "two";

$numbers[2] = "three";
```

```
$numbers[3] = "four";

$numbers[4] = "five";


foreach( $numbers as $value )

{

   echo "Value is $value <br />";

}

?>

</body>
</html>
```

This will produce the following result:

Value is 1

Value is 2

Value is 3

Value is 4

Value is 5

Value is one

Value is two

Value is three

Value is four

Value is five

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative array inside double quote while printing, otherwise it wouldnot return any value.

Example

```
<html>

<body>
```

```
<?php

/* First method to associate create array. */
```

```
$salaries = array(
                  "mohammad" => 2000,
                  "qadir" => 1000,
                  "zara" => 500
                 );

echo "Salary of mohammad is ". $salaries['mohammad'] .
"<br />"; echo "Salary of qadir is ". $salaries['qadir'].
"<br />"; echo "Salary of zara is ". $salaries['zara'].
"<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] .
"<br />"; echo "Salary of qadir is ". $salaries['qadir'].
"<br />"; echo "Salary of zara is ". $salaries['zara'].
"<br />"; ?>

</body>
</html>
```

This will produce the following result:

Salary of mohammad is 2000

Salary of qadir is 1000

Salary of zara is 500

Salary of mohammad is high

Salary of qadir is medium

Salary of zara is low

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example, we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>

<body>

<?php

   $marks = array(

            "mohammad" => array

            (

            "physics" => 35,

            "maths" => 30,

            "chemistry" => 39

            ),

            "qadir" => array

                 (

                 "physics" => 30,

                 "maths" => 32,

                 "chemistry" => 29

                 ),

                 "zara" => array

                 (
```

```
                "physics" => 31,

                "maths" => 22,

                "chemistry" => 39

                )

        );

    /* Accessing multi-dimensional array
    values */ echo "Marks for mohammad in
    physics : " ; echo $marks['mohammad']
    ['physics'] . "<br />"; echo "Marks for
    qadir in maths : ";

    echo $marks['qadir']['maths'] . "<br
    />"; echo "Marks for zara in chemistry :
    " ; echo $marks['zara']['chemistry'] .
    "<br />";

?>

</body>

</html>
```

This will produce the following result:

```
Marks for mohammad in physics : 35

Marks for qadir in maths : 32

Marks for zara in chemistry : 39
```

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other non-alphanumeric characters are replaced with a hexadecimal values. After the information is encoded, it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send up to 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out the following example by putting the source code in test.php script.

```
<?php

   if( $_GET["name"] || $_GET["age"] )

   {

      echo "Welcome ". $_GET['name'].
      "<br />"; echo "You are ". $_GET['age'].
      " years old.";
```

```
      exit();
   }
?>
<html>
<body>
   <form action="<?php $_PHP_SELF ?>"
   method="GET"> Name: <input
   type="text" name="name" /> Age: <input
   type="text" name="age" />
   <input type="submit" />
   </form>
</body>
</html>
```

It will produce the following result:

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out the following example by putting the source code in test.php script.

```
<?php

  if( $_POST["name"] || $_POST["age"] )

  {

      echo "Welcome ". $_POST['name'].
      "<br />"; echo "You are ". $_POST['age'].
      " years old."; exit();

  }

?>

<html>

<body>

  <form action="<?php $_PHP_SELF ?>" method="POST">


  Name: <input type="text" name="name" />

  Age: <input type="text" name="age" />


  <input type="submit" />

  </form>

</body>

</html>
```

It will produce the following result:

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out the following example by putting the source code in test.php script.

```
<?php

  if( $_REQUEST["name"] || $_REQUEST["age"] )

  {

      echo "Welcome ". $_REQUEST['name']. "<br
      />"; echo "You are ". $_REQUEST['age']. "
      years old."; exit();

  }

?>

<html>

<body>

  <form action="<?php $_PHP_SELF ?>" method="POST">


  Name: <input type="text" name="name" />

  Age: <input type="text" name="age" />


  <input type="submit" />

  </form>

</body>

</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

It will produce the following result:

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required, then instead of changing thousands of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file, then the **include()** function generates a warning but the script will continue execution.

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file, then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try the following two examples where file does not exist, then you will get different results.

```
<html>
```

```
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce the following result:

```
This is an example to show how to include wrong PHP file!
```

Now let us try same example with require() function.

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This time file execution halts and nothing is displayed.

NOTE: You may get plain warning messages or fatal error messages or nothing at all. Thisdepends on your PHP Server configuration.

File Handling

This chapter will explain the following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a file

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.

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If the file does not exist, then it attempts to create a file.
w+	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 the file does not exist, then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.

If an attempt to open a file fails, then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

,

After making a changes to the opened file it is important to close it with the **fclose()**function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

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 file's 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.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable and then displays those contents on the web page.

```
<html>

<head>

<title>Reading a file using PHP</title>

</head>

<body>
```

```php
<?php

$filename = "/home/user/guest/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>
```

It will produce the following result:

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 will stop after the specified length has been reached.

The following example creates a new text file and then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exist()** 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

if( file_exist( $filename ) )

{

    $filesize = filesize( $filename );

    $msg = "File created with name
    $filename "; $msg .= "containing
    $filesize bytes"; echo ($msg );

}

else

{

    echo ("File $filename does not exit" );

}

?>
</body>

</html>
```

It will produce the following result:

```
Error in opening new file
```

We have covered all the function related to file input and out in the PHP File SystemFunctionchapter.

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

Creating PHP Function

It is very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it.

The following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<html>

<head>

<title>Writing PHP Function</title>

</head>

<body>


<?php

/* Defining a PHP Function */

function writeMessage()

{

   echo "You are really a nice person, Have a nice time!";
```

```
}

/* Calling a PHP Function */
```

```
writeMessage();

?>

</body>

</html>
```

This will display the following result:

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. The following example takes two integer parameters and adds them together and then prints them.

```
<html>

<head>

<title>Writing PHP Function with Parameters</title>

</head>

<body>


<?php

function addFunction($num1, $num2)
```

```
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</html>
```

This will display the following result:

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

The following example depicts both the cases.

```
<html>

<head>

<title>Passing Argument by Reference</title>
</head>

<body>
<?php

function addFive($num)

{
   $num += 5;

}


function addSix(&$num)
{

   $num += 6;

}
$orignum = 10;

addFive( &$orignum );
echo "Original Value is $orignum<br />";

addSix( $orignum );
echo "Original Value is $orignum<br />";

?>

</body>
```

```
</html>
```

This will display the following result:

```
Original Value is 10

Original Value is 16
```

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

The following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>

<head>

<title>Writing PHP Function which returns value</title>

</head>

<body>


<?php

function addFunction($num1, $num2)

{

  $sum = $num1 + $num2;
```

```
   return $sum;

}

$return_value = addFunction(10, 20);

echo "Returned value from the function : $return_value";

?>

</body>

</html>
```

This will display the following result:

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

The following function prints NULL in case use does not pass any value to this function.

```
<html>

<head>

<title>Writing PHP Function which returns value</title>

</head>

<body>
```

```
<?php

function printMe($param = NULL)

{

    print $param;

}

printMe("This is test");

printMe();

?>


</body>

</html>
```

This will produce the following result:

```
This is test
```

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:

- Server script sends a set of cookies to the browser. For example, name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- Next time, when the browser sends any request to the web server, it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

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 up to 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 variablecalled HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value -**This sets the value of the named variable and is the content that youactually want to store.
- **Expiry -** This specify 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 forwardslash character permits the cookie to be valid for all directories.

- **Domain -** This can be used to specify the domain name in very large domains andmust 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 bysecure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

The following example will create two cookies **name** and **age**. These cookies will expire after an 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. The simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. The 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["name"] . "<br />";

?>

</body>

</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>

<head>

<title>Accessing Cookies with PHP</title>

</head>

<body>

<?php

  if( isset($_COOKIE["name"]))

    echo "Welcome " . $_COOKIE["name"] . "<br />";else
```

```
    echo "Sorry... Not recognized" . "<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>
```

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Bore using any session variable make sure you have setup this path.When a session is started, the following actions take place:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if

none is started, then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session and then registers a variable called **counter** that is incremented each time the page is visited during the session.

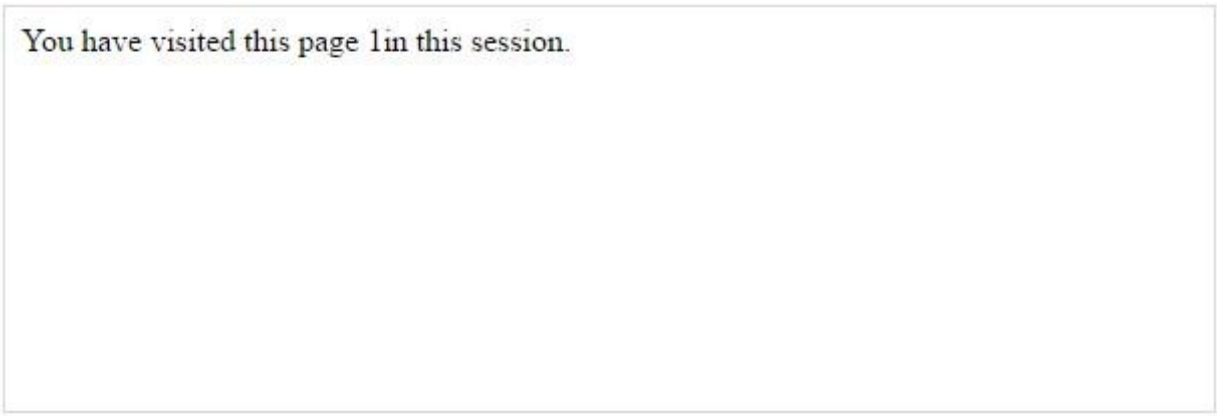
Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result:

```
<?php
   session_start();
   if( isset( $_SESSION['counter'] ) )
   {
      $_SESSION['counter'] += 1;
   }
   else
   {
      $_SESSION['counter'] = 1;
   }
   $msg = "You have visited this page ".
   $_SESSION['counter']; $msg .= "in this session.";
?>
```

```
<html>

<head>

<title>Setting up a PHP session</title>

</head>

<body>

<?php      echo ( $msg ); ?>

</body>

</html>
```

It will produce the following result:



Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable, then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable:

```
<?php

   unset($_SESSION['counter']);

?>
```

Here is the call which will destroy all the session variables:

```
<?php

   session_destroy();

?>
```

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

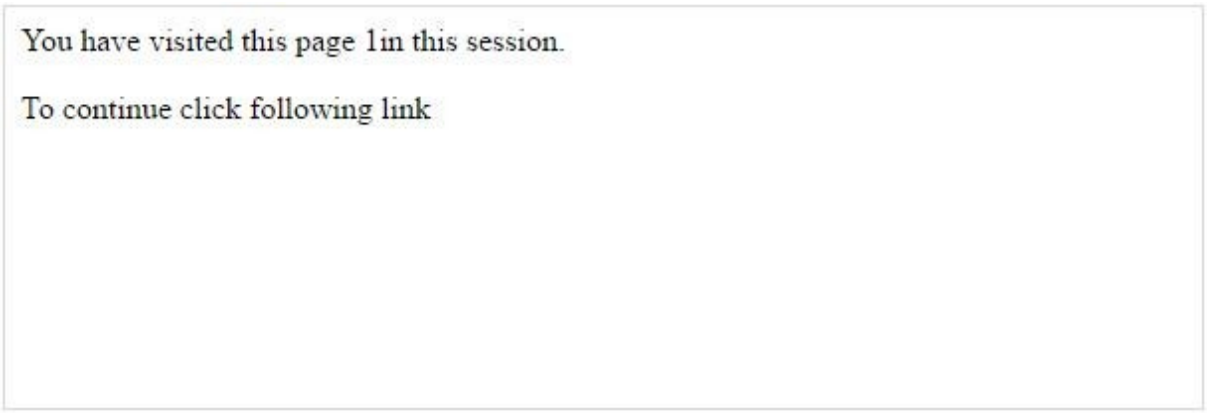
Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php

   session_start();



   if (isset($_SESSION['counter'])) {

      $_SESSION['counter'] = 1;

   } else {

      $_SESSION['counter']++;

   }

?>

   $msg = "You have visited this page ". $_SESSION['counter'];
```

```
   $msg .= "in this session.";

   echo ( $msg );

<p>

To continue click following link <br />

<a href="nextpage.php?<?php echo
htmlspecialchars(SID); >"></p>
```

It will produce the following result:



File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file**php.ini**

The process of uploading a file follows these steps:

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text field, then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual, while writing files, it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only, then process will fail.

An uploaded file could be a text file or image file or any document.

Creating an Upload Form

The following HTML code creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

Creating an upload script

There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create the following five variables:

- **$_FILES['file']['tmp_name']-** the uploaded file in the temporary directory onthe web server.
- **$_FILES['file']['name'] -** the actual name of the uploaded file.
- **$_FILES['file']['size'] -** the size in bytes of the uploaded file.

- **$_FILES['file']['type'] -** the MIME type of the uploaded file.
- **$_FILES['file']['error'] -** the error code associated with this file upload.

```
<!DOCTYPE html>
<html>
	<head>
		<title>FILE UPLOADING</title>
	</head>
	<body>
		<form method="post" action="#"
enctype="multipart/form-data">
			<pre>
				<label>SELECT FILE TO UPLOAD</label>
				<input type="file" name="t1" required>

				<input type="submit" name="btn"
value="UPLOAD">

			</pre>
		</form>
		<?php
			if(isset($_POST["btn"]))
			{
				$filename=$_FILES["t1"]["name"];
				$tmpname=$_FILES["t1"]["tmp_name"];
				$type=$_FILES["t1"]["type"];
				$size=$_FILES["t1"]["size"];
				echo $size;
				if($type=="image/jpeg")
				{
					if(!is_dir("uploads"))
						mkdir("uploads");

move_uploaded_file($tmpname,"C:/xampp/htdocs/php26thaug/uploads/".
$filename);
				}
				else
					echo "<script>alert('File format not
supported!')</script>";
```

```
                        }
                        ?>
                </body>
        </html>
```

Data Handling with MYSQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

The commonly used operations are given below:

- **Select or View records**
  - Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function **mysqli_query.**

To View Records from a database table

```
<?php
$con=mysqli_connect("localhost","root","","shop");
//localhost-server name
//root-mysql user name
// " "-empty password
//shop-database name
if(!$con)
        die("cannot connect to database");
else
{
        $query="select * from product";
        $rs=mysqli_query($con,$query);
        if(mysqli_num_rows($rs)>0)
        {
                echo "<table border='1px' align='center'
                cellpadding='10px' cellspacing='0px'>";
                echo "<tr><th>PRODUCT ID</th>";
                echo "<th>PRODUCT NAME</th>";
                echo "<th>PRODUCT DESCRIPTION</th>";
                echo "<th>PRODUCT COST</th></tr>";
                while($row=mysqli_fetch_array($rs))
                {
                        echo "<tr><td>$row[0]</td>";
                        echo "<td>$row[1]</td>";
                        echo "<td>$row[2]</td>";
                        echo "<td>$row[3]</td></tr>";
```

```
            }
            echo "</table>";
      }}
?>
```

- ## Insert records
  - Data can be updated into MySQL tables by executing SQL INSERT statement through PHP function **mysqli_query**.
- Below is a simple example to insert records into **product** table

To Add a record to a database table

```
<!DOCTYPE html>
<html>
	<head>
		<title>ADD A RECORD TO A DATABASE TABLE</title>
	</head>
	<body>
		<form method="post" action="#">
			<pre>
				<label>PRODUCT ID</label>
				<input type="text" name="t1">

				<label>PRODUCT NAME</label>
				<input type="text" name="t2">

				<label>PRODUCT DESCRIPTION</label>
				<input type="text" name="t3">

				<label>PRODUCT COST</label>
				<input type="number" name="t4">

				<input type="submit" name="btn"
				value="ADD PRODUCT">
			</pre>
		</form>
		<?php
			if(isset($_POST["btn"]))
			{
				$id=$_POST["t1"];
				$name=$_POST["t2"];
				$des=$_POST["t3"];
				$cost=$_POST["t4"];
```

```
                        include("connect.php");



                        $sql="insert into product(pid,pname,pdes,pcost)
                        values('".$id."','".$name. "','". $des. "',". $cost.")";
                        $n=mysqli_query($con,$sql);
                        if($n==1)
                        {
                                echo "<script>alert('Product added!')</script>";
                        echo
"<script>window.location.href='showproducts.php'</script>";
                        }
                }


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

Delete records

- Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysqli_query**.
- Following is a simple example to delete records into **product** table. To delete a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in product table.

To Delete a record from a database table

```
<!DOCTYPE html>
<html>
<head>
<title>DELETE A RECORD FROM A DATABASE TABLE</title>
</head>
<body>
<form method="post" action="#">
<pre>
<label>SELECT PRODUCT ID</label>
<select name="t1">
```

```
	<?php

	include("connect.php");
	$sql="select * from product";
	$rs=mysqli_query($con,$sql);
	while($row=mysqli_fetch_array($rs))
{
		?>
	<option value="<?=$row[0]?>">
	<?=$row[0]?>
	</option>
	<?php
	}
	?>
	</select>

	<input type="submit" name="btn"
	value="DELETE PRODUCT">
	</pre>
	</form>
	<?php
	if(isset($_POST["btn"]))
	{
	$id=$_POST["t1"];
	$query="delete from product where pid='". $id. "'";
	$n=mysqli_query($con,$query);
	if($n==1)
	{
	echo "<script>alert('product deleted')</script>";
	echo "<script>window.location.href='del_product.php'</script>";
	}
	}
	?>
	</body>
</html>
```

- ## Update records
  - Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysqli_query**.
  - Below is a simple example to update records into **product** table. To update a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in product table.

Update data in database table

```
<!DOCTYPE html>
<html>
	<head>
		<title>UPDATE A RECORD IN A DATABASE TABLE</title>
	</head>
	<body>
		<form method="post" action="#">
			<label>SELECT PRODUCT ID</label>
			<select name="t1">
			<?php
				include("connect.php");
				session_start();
				$sql="select pid from product";
				$rs=mysqli_query($con,$sql);
				while($row=mysqli_fetch_array($rs))
				{
					?>
				<option value="<?=$row[0]?>">
				<?=$row[0]?>
				</option>
				<?php
				}
			?>
		</select>
		<input type="submit" value="SHOW PRODUCT" name="btn">

	<?php
		if(isset($_POST["btn"]))
		{
```

```
                $id=$_POST["t1"];
                $_SESSION["pid"]=$id;
                $query="select * from product where pid='". $id. "'";
                $rs=mysqli_query($con,$query);
                if(mysqli_num_rows($rs)==1)
                {
                        $row=mysqli_fetch_array($rs);
                        echo "<table border='1px' align='center'
                        cellpadding='10px' cellspacing='0px'>";
                        echo "<tr><th>PRODUCT ID</th>";
                        echo "<th>PRODUCT NAME</th>";
                        echo "<th>PRODUCT DESCRIPTION</th>";
                        echo "<th>PRODUCT COST</th></tr>";
                        echo "<tr><td>$row[0]</td>";
                        echo "<td><input type='text' name='t2'
                        value=$row[1]></td>";
                        echo "<td><input type='text' name='t3'
                        value=$row[2]></td>";
                        echo "<td><input type='text' name='t4'
                        value=$row[3]></td></tr>";

                        echo "<tr><td colspan='4' align='center'>
                        <input type='submit' name='btn1'
                        value='UPDATE'></td></tr>";
                }
        }
        ?>
</form>
<?php
        if(isset($_POST["btn1"]))
        {
                $id=$_SESSION["pid"];
                $name=$_POST["t2"];
                $des=$_POST["t3"];
                $cost=$_POST["t4"];
                $q="update product set pname='". $name. "',
        pdes='". $des. "',pcost=". $cost. " where pid='". $id. "'";
        $n=mysqli_query($con,$q);
                if($n==1)
                {
                        echo "<script>alert('product updated')</script>";
```

```
echo "<script>window.location.href='showproducts.php'</script>";

            }
        }
?>


        </body>
</html>
```

WORD-PRESS

WordPress is an open source **Content Management System (CMS),** which allows the users to build dynamic websites and blogs. WordPress is the most popular blogging system on the web and allows updating, customizing and managing the website from its back-end CMS and components.

WordPress Login / Dashboard

- **Login to My WordPress Site**

Go to Wordpress.org->Download wordpress and extract the zip file.
Paste the extracted folder in c: /xampp/htdocs
Rename the folder in htdocs to say "store"
Use link localhost/store to run wordpress

You will get the screen below:

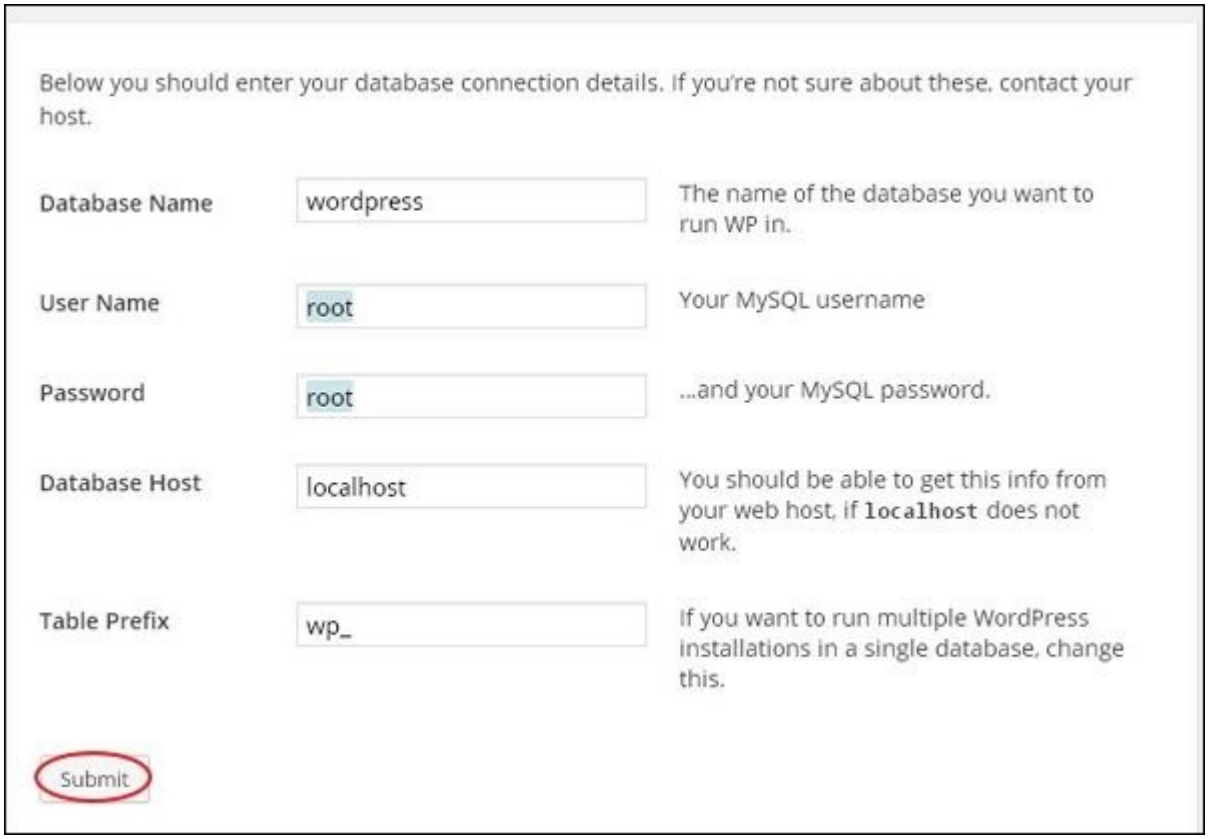
Step 1:

Step 2:


Welcome to WordPress. Before getting started, we need some information on the database. You will need to know the following items before proceeding.
1. Database name
2. Database username
3. Database password
4. Database host
5. Table prefix (if you want to run more than one WordPress in a single database)
We're going to use this information to create a wp-config.php file. If for any reason this automatic file creation doesn't work, don't worry. All this does is fill in the database information to a configuration file. You may also simply open wp-config-sample.php in a text editor, fill in your information, and save it as wp-config.php. Need more help? We got it.
In all likelihood, these items were supplied to you by your Web Host. If you do not have this information, then you will need to contact them before you can continue. If you're all ready...
Let's go!

Step 3:

Here, you have to enter theMySQL database information about the as shown below:



Step 4:

After installation and entering administrative information you will get the screen below:

use the below url:

Url for admin:
Localhost/store/wp-admin

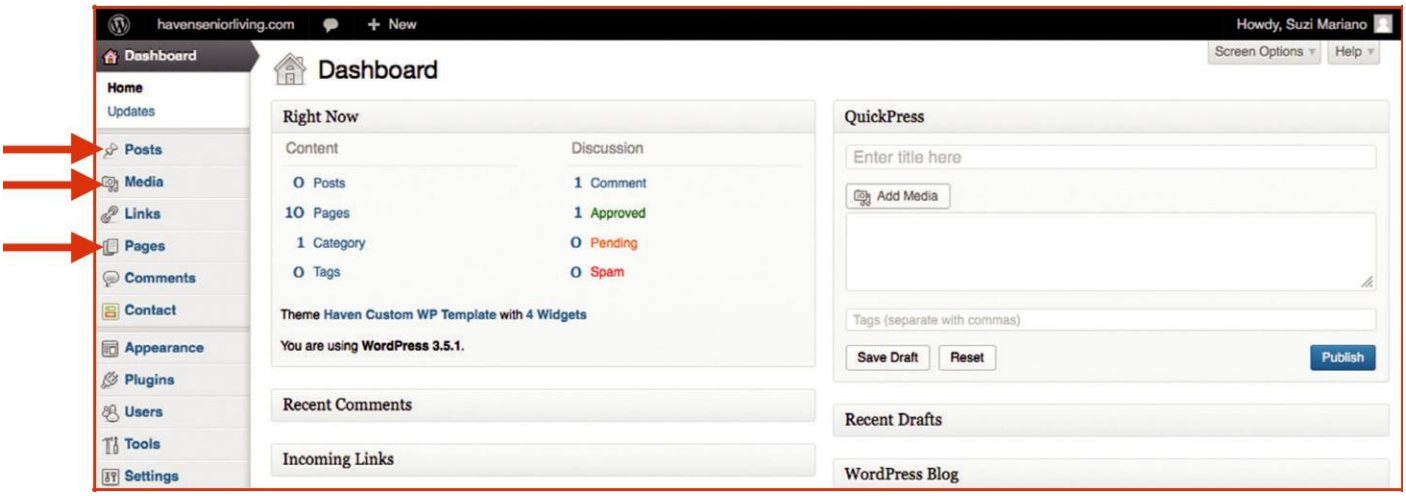
Url for general user:
Localhost/store

-
- Use the above url for admin to get to the admin login page of your WordPress website.
- Enter your login and password.
- Click “Log In”.

d. This will take you to your dashboard (the back-end of your website).

What's My "Dashboard"?

- The Dashboard is the first screen you see when you log into the administration area of your website.
- The dashboard gives you an at-a-glance overview of what's happening with your website.



- From your dashboard (see above) you can get to your:
  - Website pages.
  - Media (Pictures/Images/PDFs) you have loaded.
  - Posts you have made to your blog.

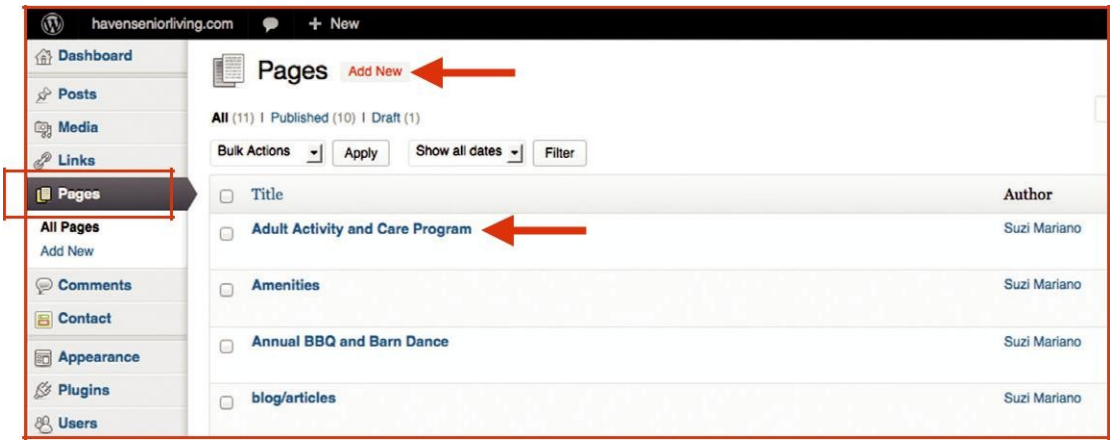
3. Pages Menu - Adding/Editing My Pages

a. Double Click "Pages" on Dashboard to bring up your Pages Menu.

b. From here you can:

1. "Add New" page (click orange "Add New" button").

2. Edit a current page (click on any other page name).



4. Adding a New Page to My Website

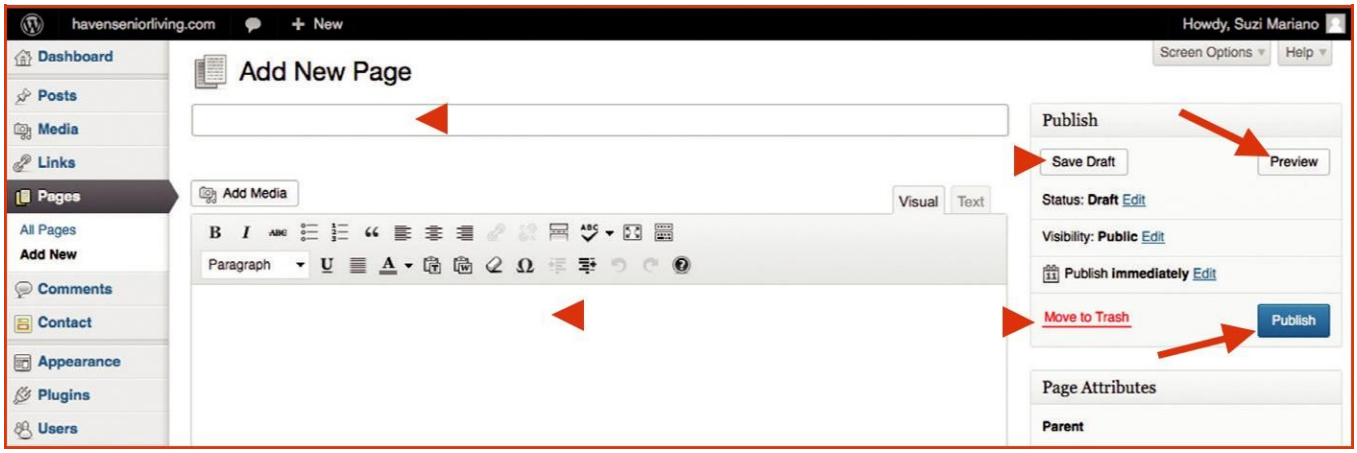
a. Click "Add New" from Pages Menu (see above).

b. Your new page will look like the below.

c. Add your content:

1. Web page Name/Headline.

2. Web page Text/Body Copy.



3. Click "Preview" to preview the content you have put on the page (NOTE: this does not mean it has been saved – to save you must click the "Publish" button).
4. Click "Save Draft" to save a draft of your page if you are still working, but are not ready for the page to be published to your website.
5. Click "Publish" to make a new page of your website live.
6. Click "Move to Trash" to delete your new page.

5. Editing the Content on My Website (kind of like Microsoft Word)

a. See diagram below to edit content on a page of your website:

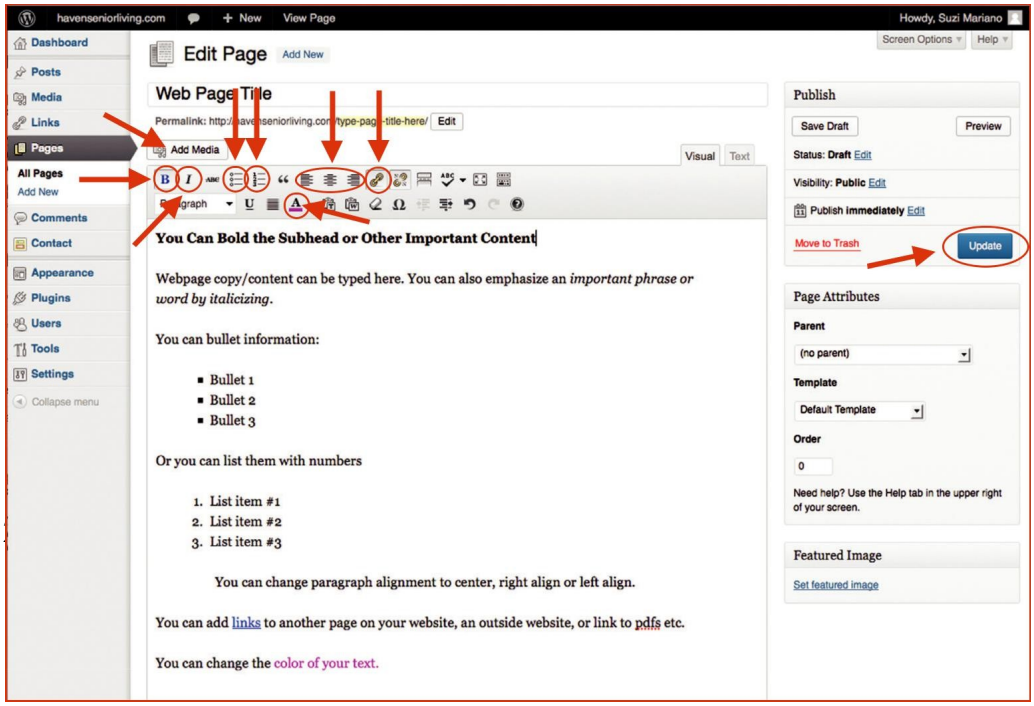
1. **Bold:** You Can Bold the Subhead or Other Important Content(Highlight your text first, click "B" button).
2. **Italicize:** You can italicize a word or phrase(Highlight your text first, click "I" button).
3. **Bullet List:** You can bullet information(Highlight your text first, click bullet icon).
4. **Numeric List:** Or you can list them with numbers(Highlight your text first, click # list icon).
5. **Alignment:** You can change paragraph alignment to center, right align or left align(Highlight your text

   first, click paragraph line icon).

6. **Links:** You can add links to another page on your website or an outside website.

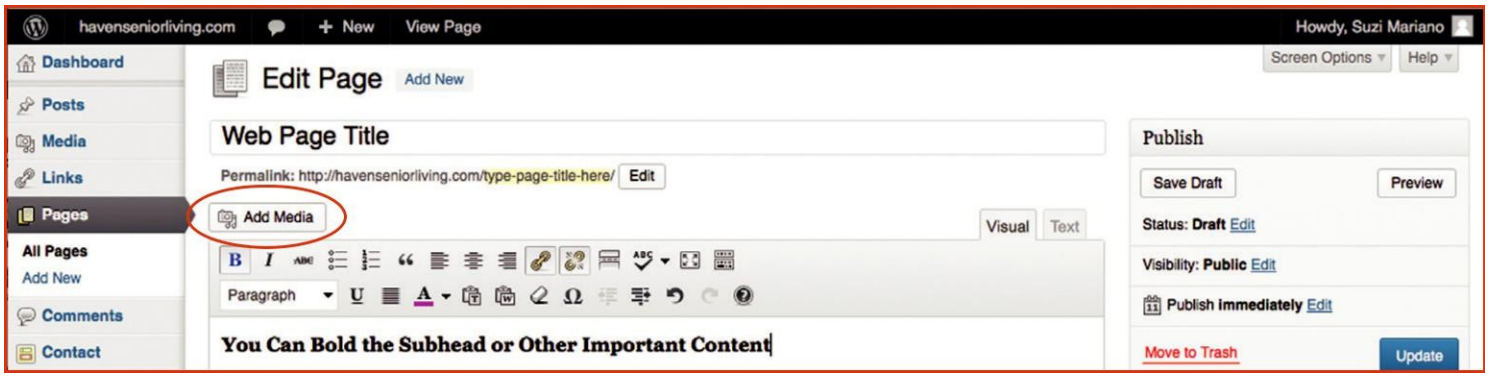
   (Highlight your text first, click link icon) (See FAQ #8 for more information).

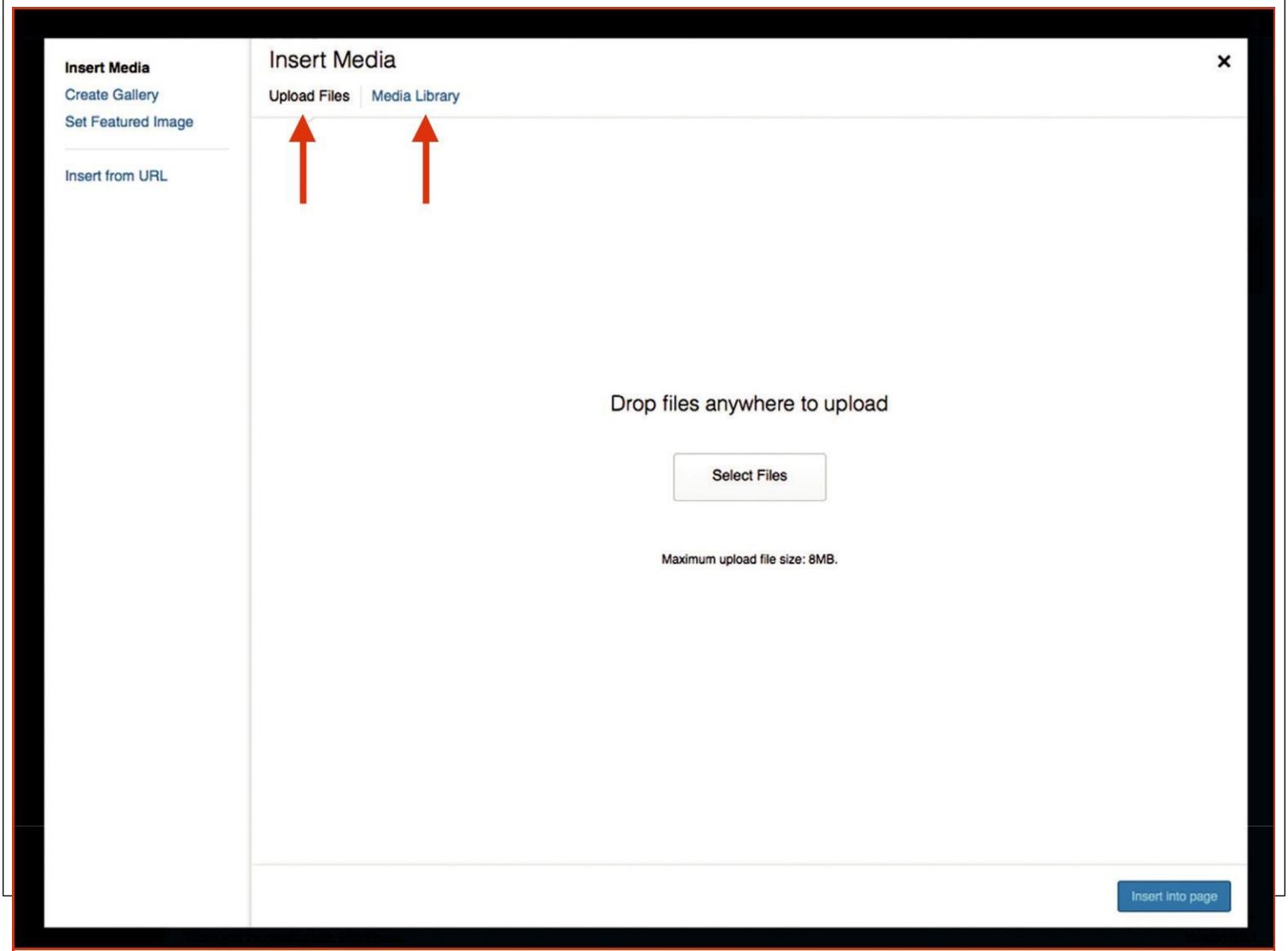
7. **Color Text:** You can change the color of your text(Highlight your text first, click color "A" icon).
8. **Add Images:** You can add an image to your website.(Add media button) (See FAQ #6 and #7 for moreinformation).
9. **ALWAYS Update:** If you make **ANY** changes to your website page, and want them to be saved - make sure to click "Update".



6. Adding an Image on My Website

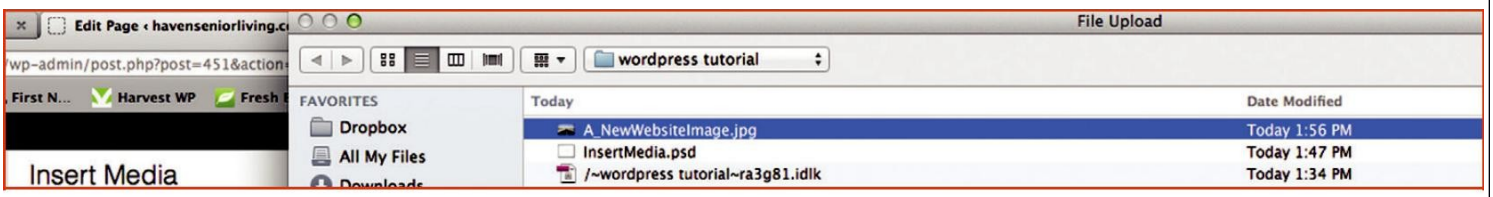
a. Click the "Add Media" Button.

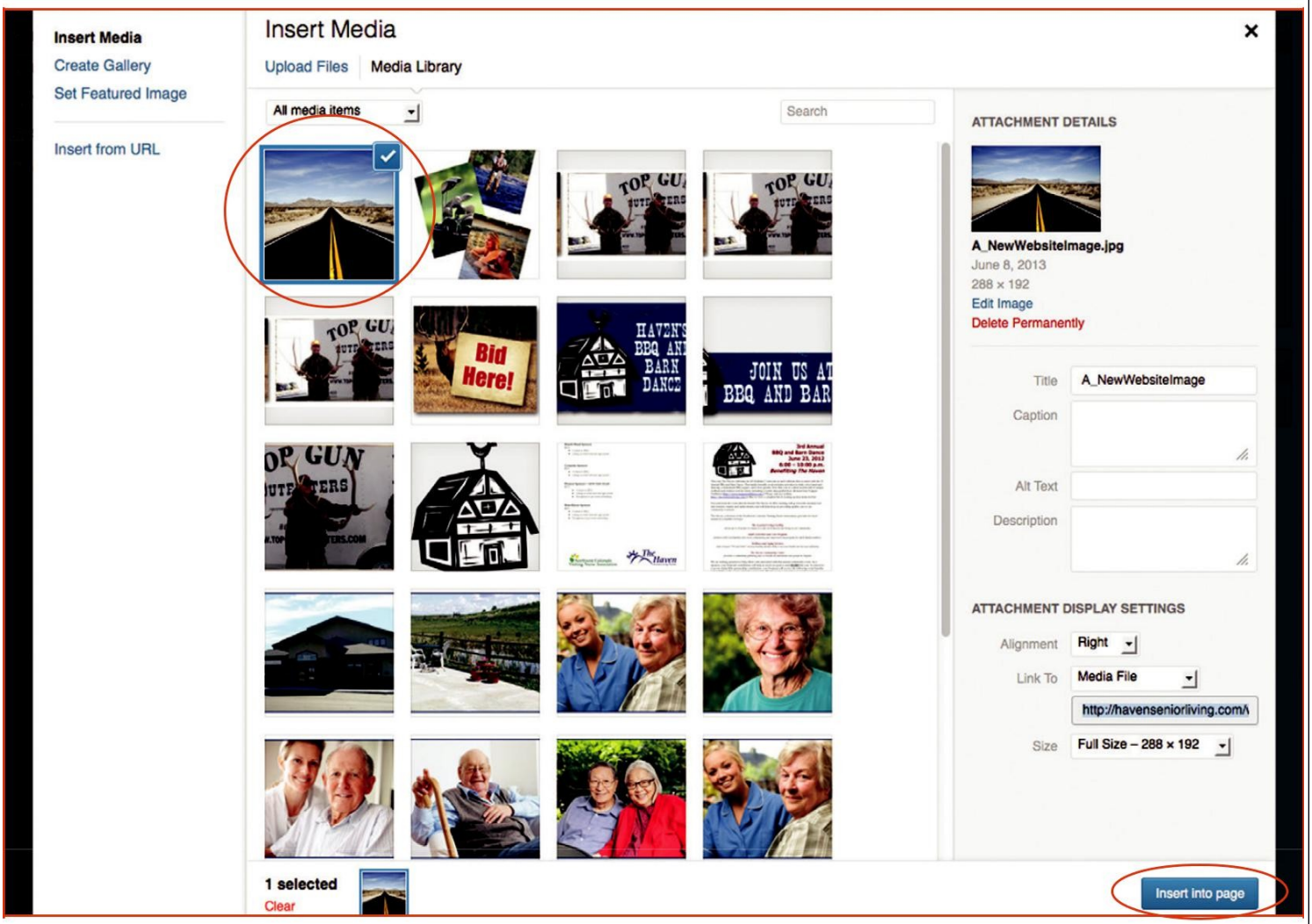




To upload a **new image**, click “Select Files”.

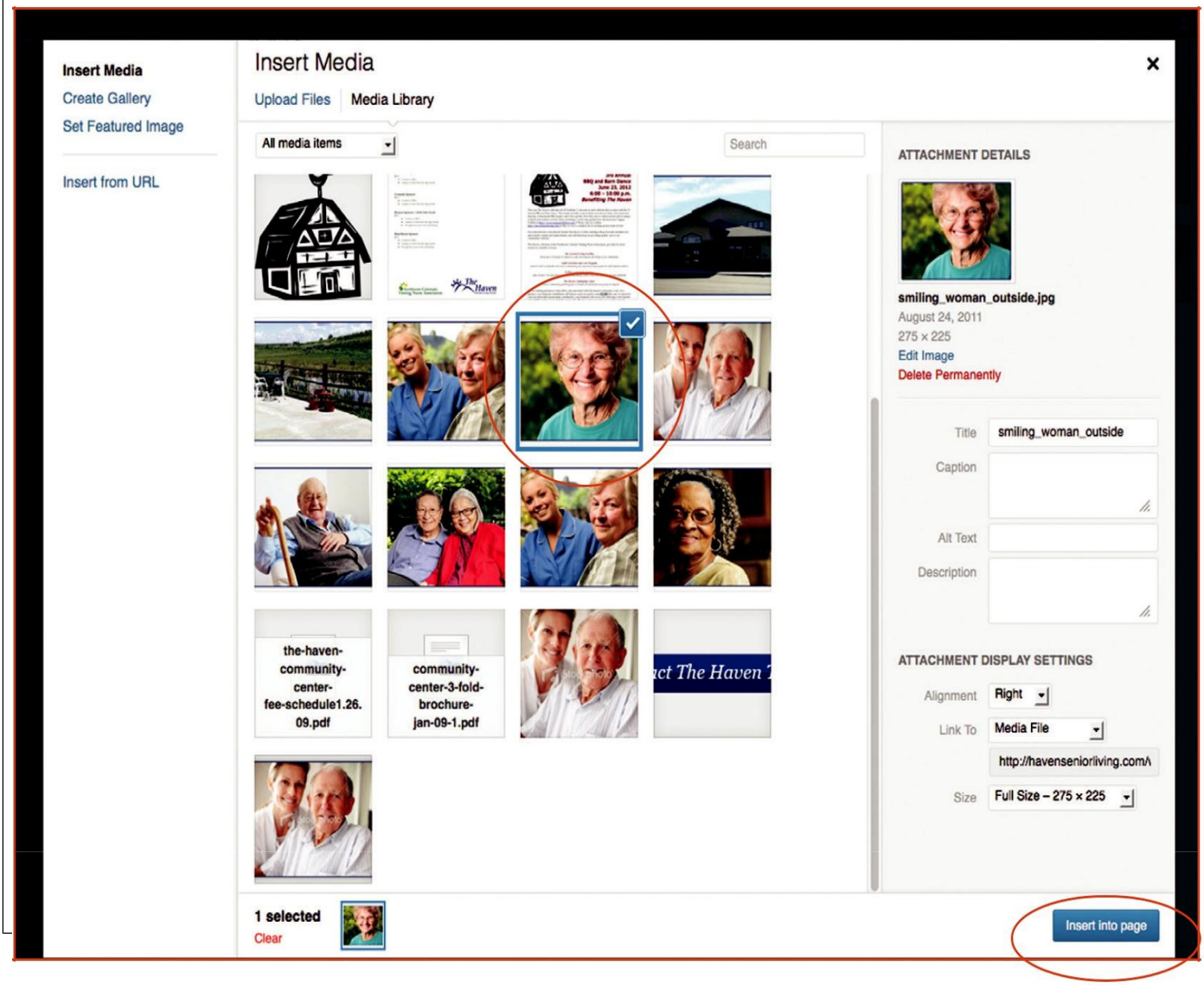
1. A new window will pop-up - Select the image you would like to add - Click “Open”.



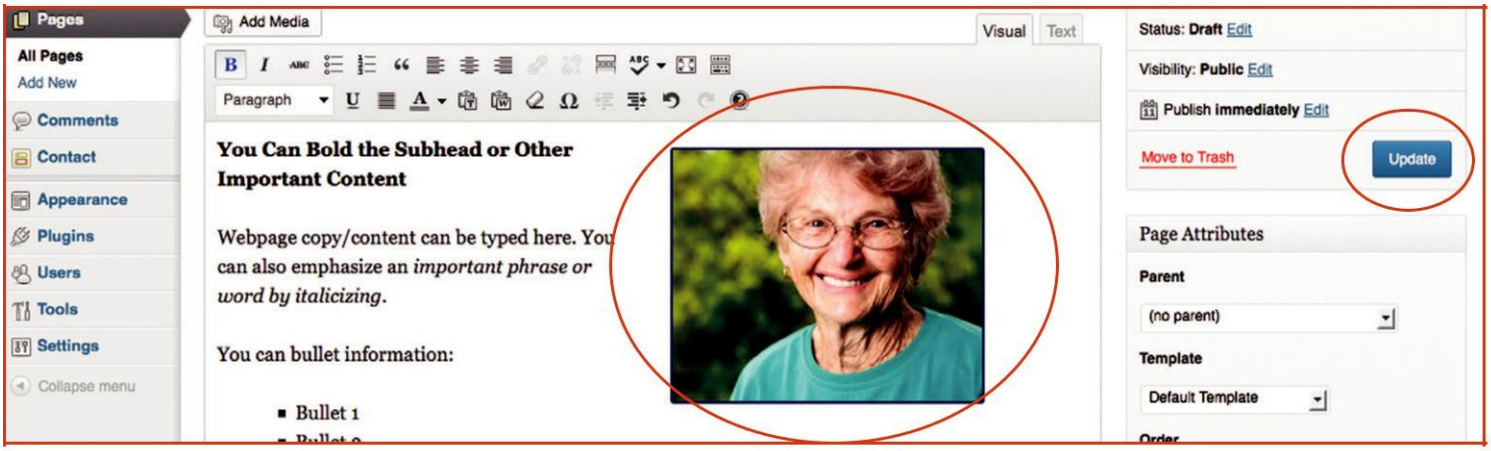


3. Your image has been added to your web page - Remember to click update to save any additions.

e. To add an image from the "**Media Library**", click on "Media Library".



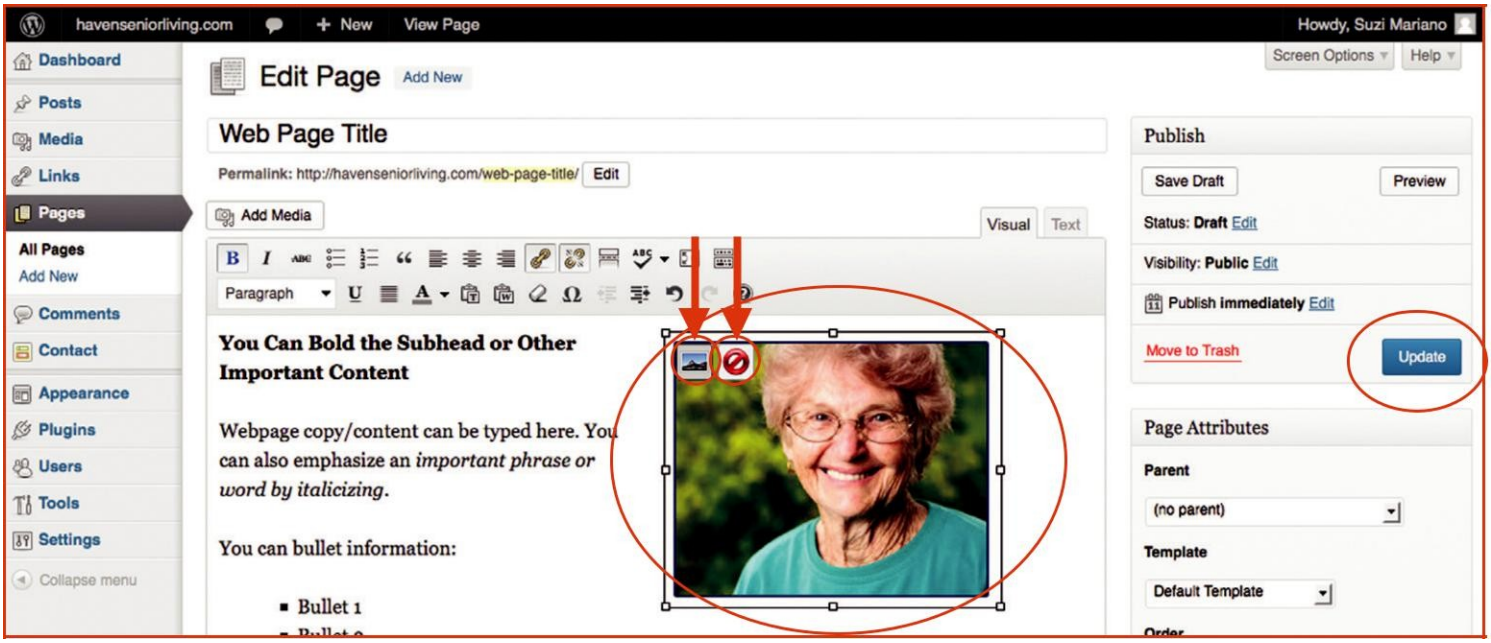
3. Your image will be added.
4. Click “Update” to save your changes.



7. Editing an Image on My Website

a. Click on your Image.

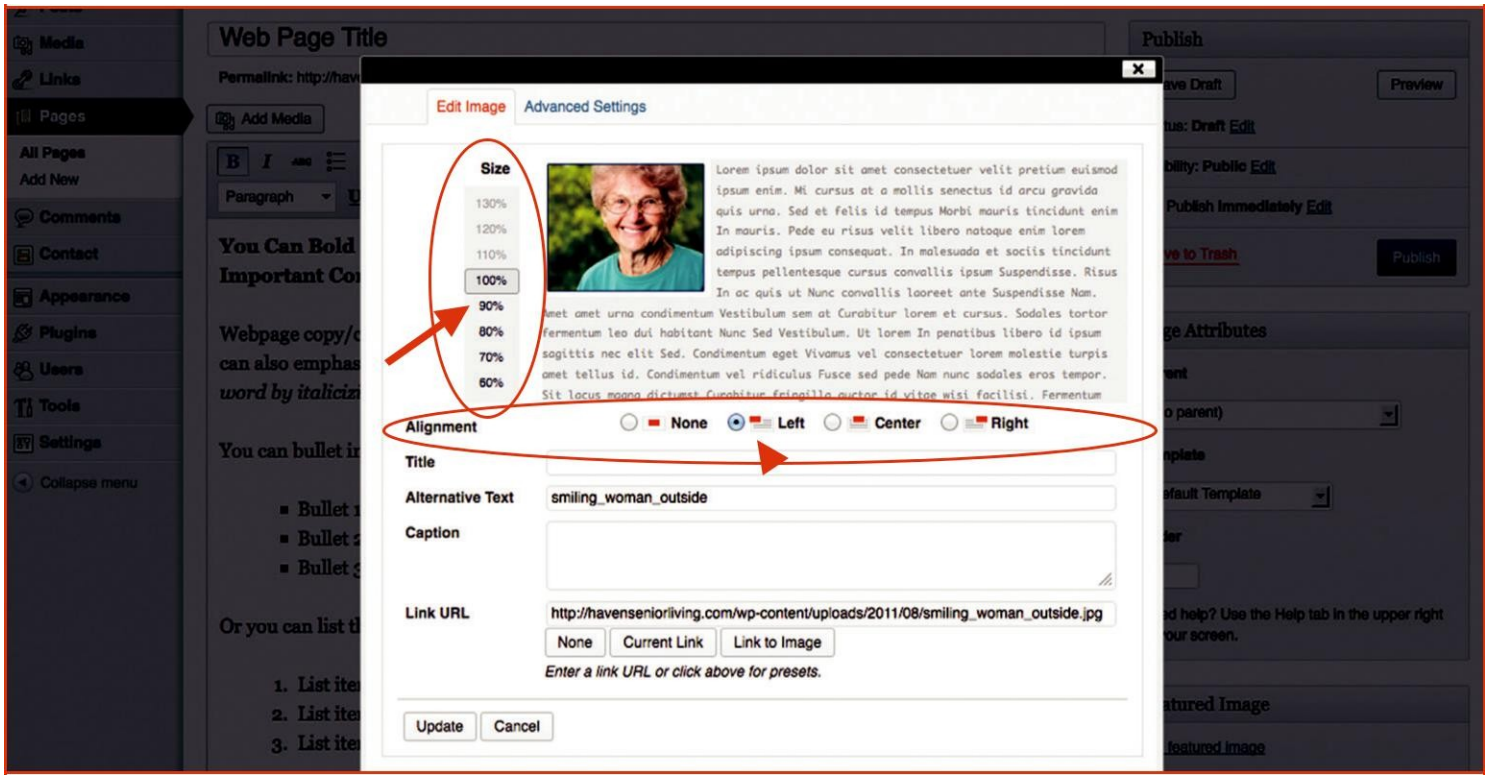
1. Click on Image icon (looks like a landscape).



a. The “Edit Image” menu will pop-up and you can change the:

   1. Size of your image.
   2. The alignment of your image.
   3. When finished, click “Update”.

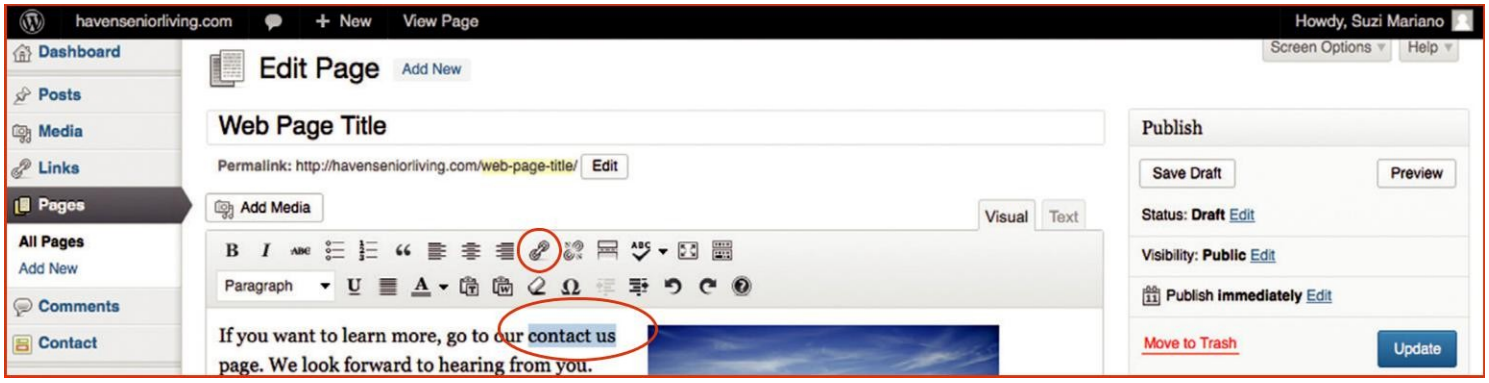
2. If you decide to “Delete” your new image, click on red delete icon.



8. Adding Links to My Content on My Website

a. Highlight text you want to link.

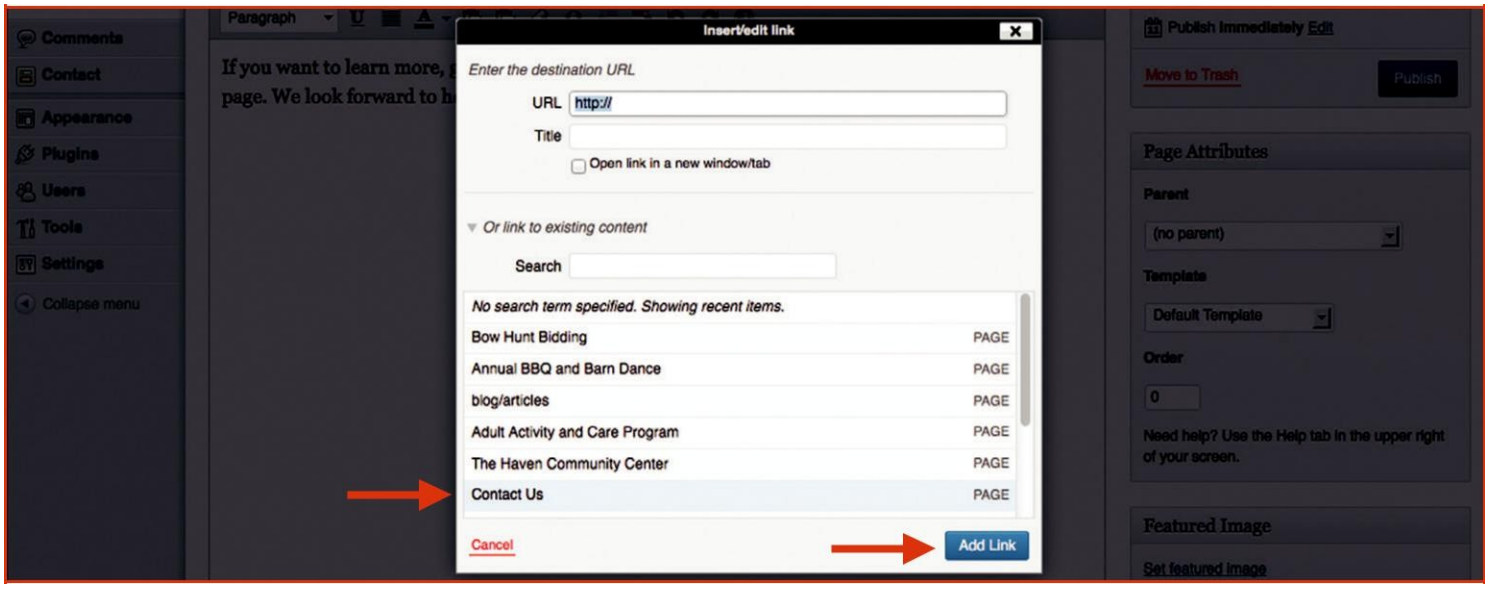
b. Click link icon.



c. A new window will pop-up.

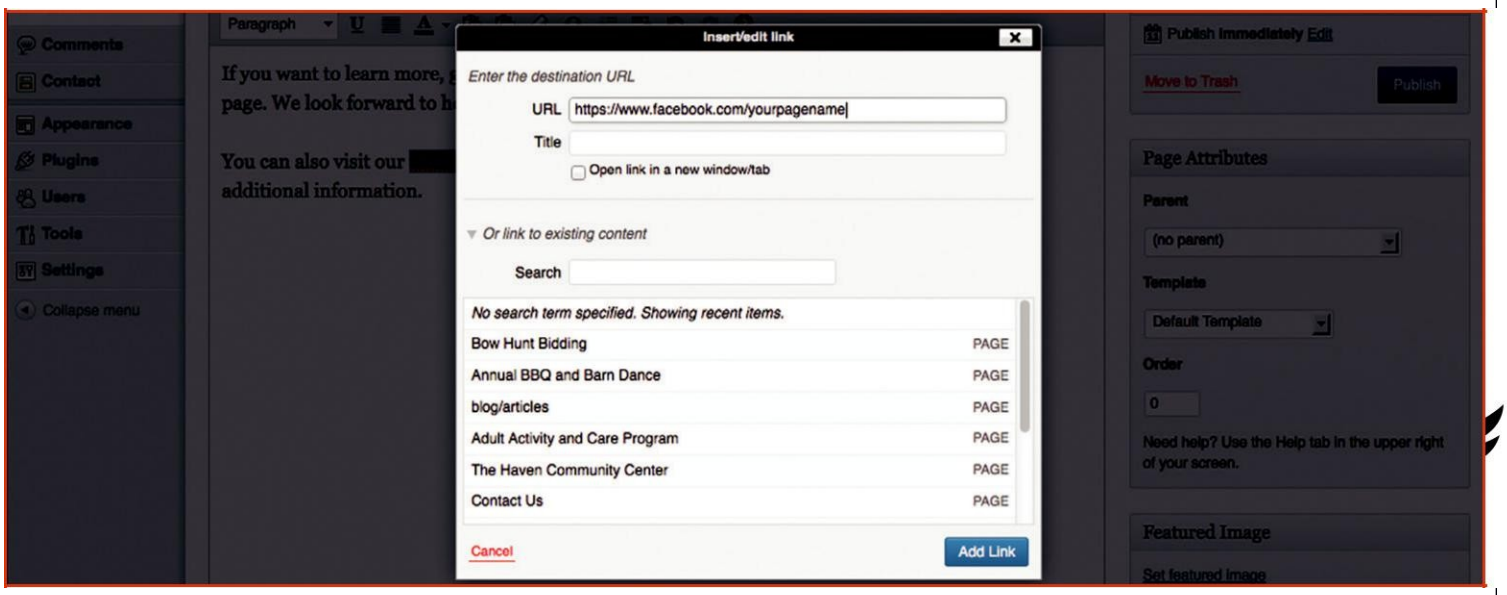
d. To link to a page **internally** (one that is already on your website):

1. Click on the page name from the listing (in this case "Contact Us").
2. Click "Add Link".



e. To link to a page **externally** (one that is not on your site):

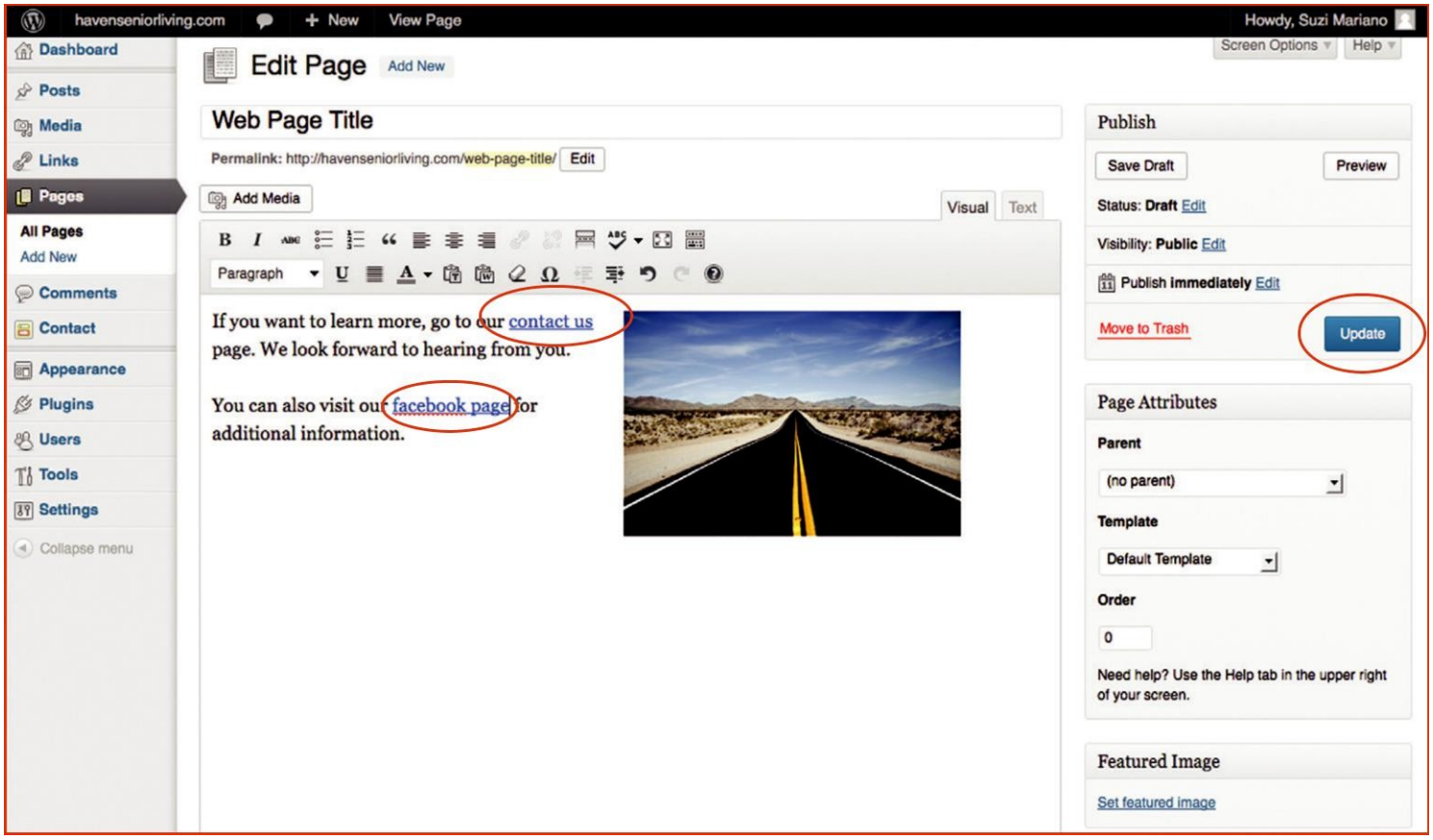
1. Type in URL of outside site within text box that says “URL”.
2. Click “Add Link”.
3. Click “Open link in a new window/tab”.



8. Adding Links to My Content - Continued

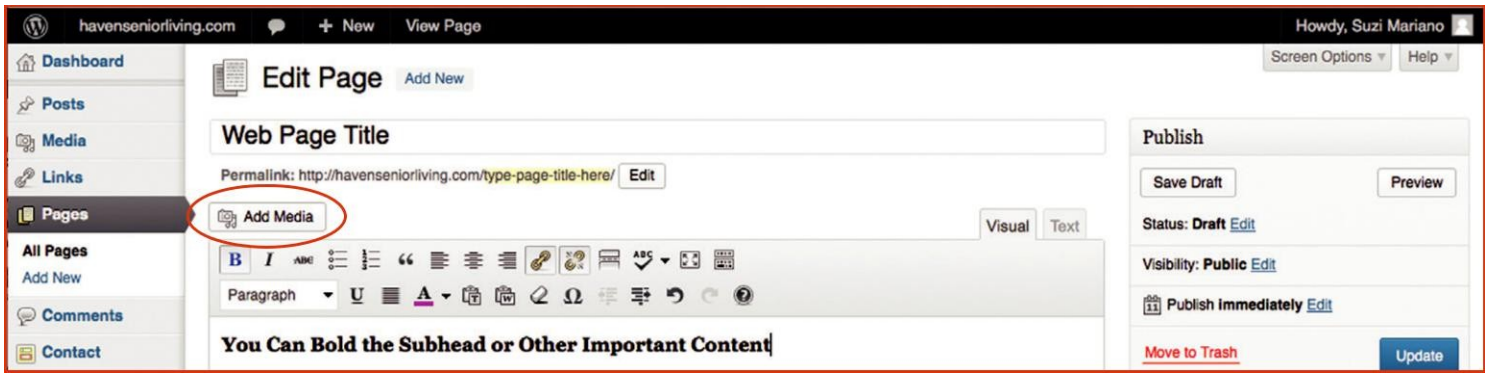
a. Your Links will be added (see below).

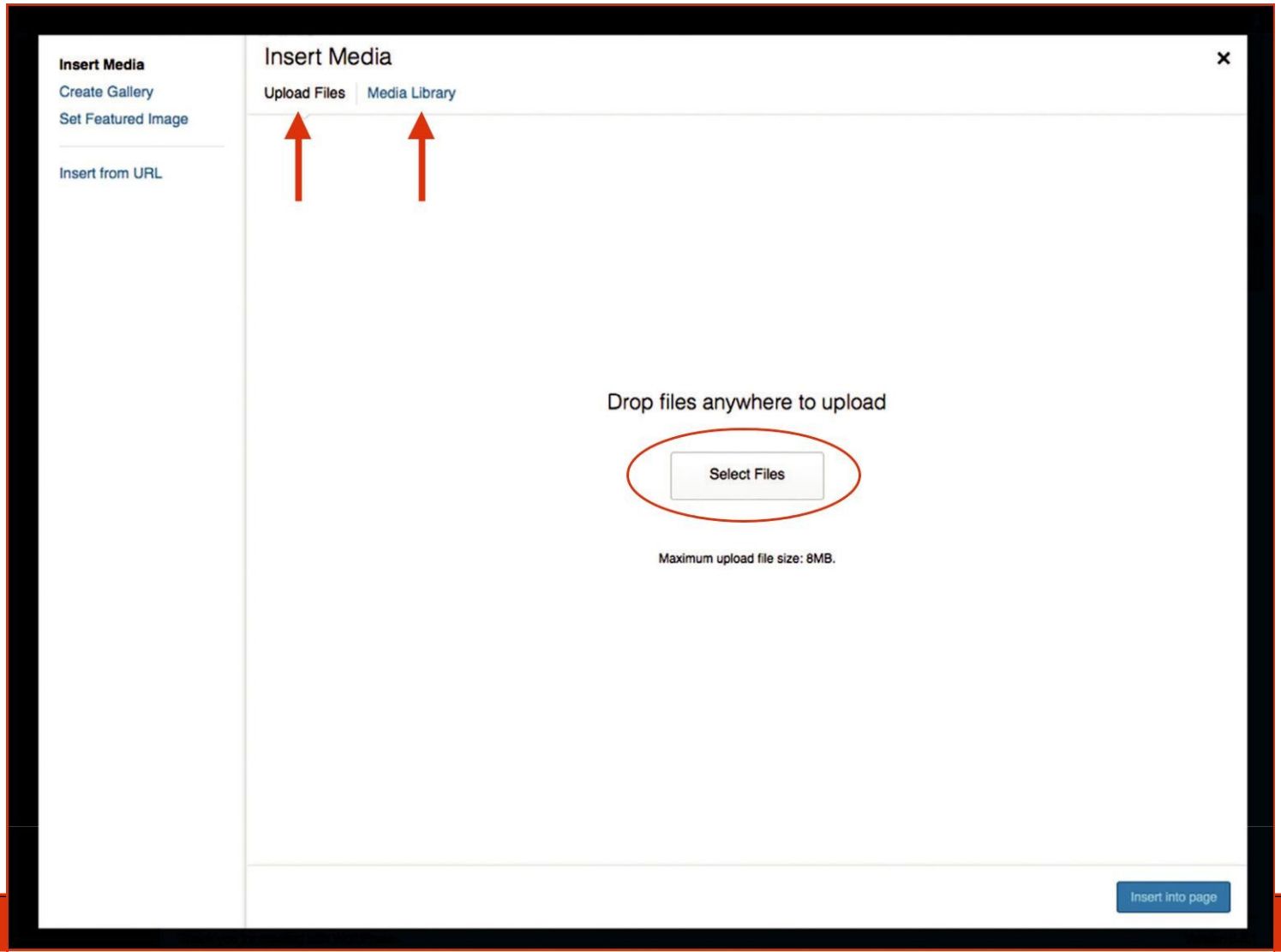
b. Remember to hit update to save your changes.



9. Adding Menus

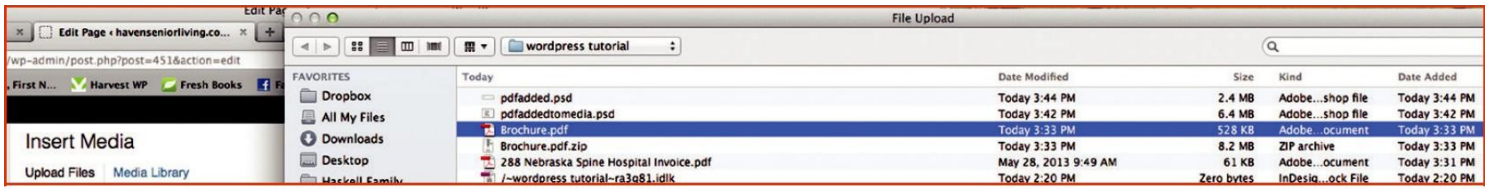
a. Click the “Add Media” Button.

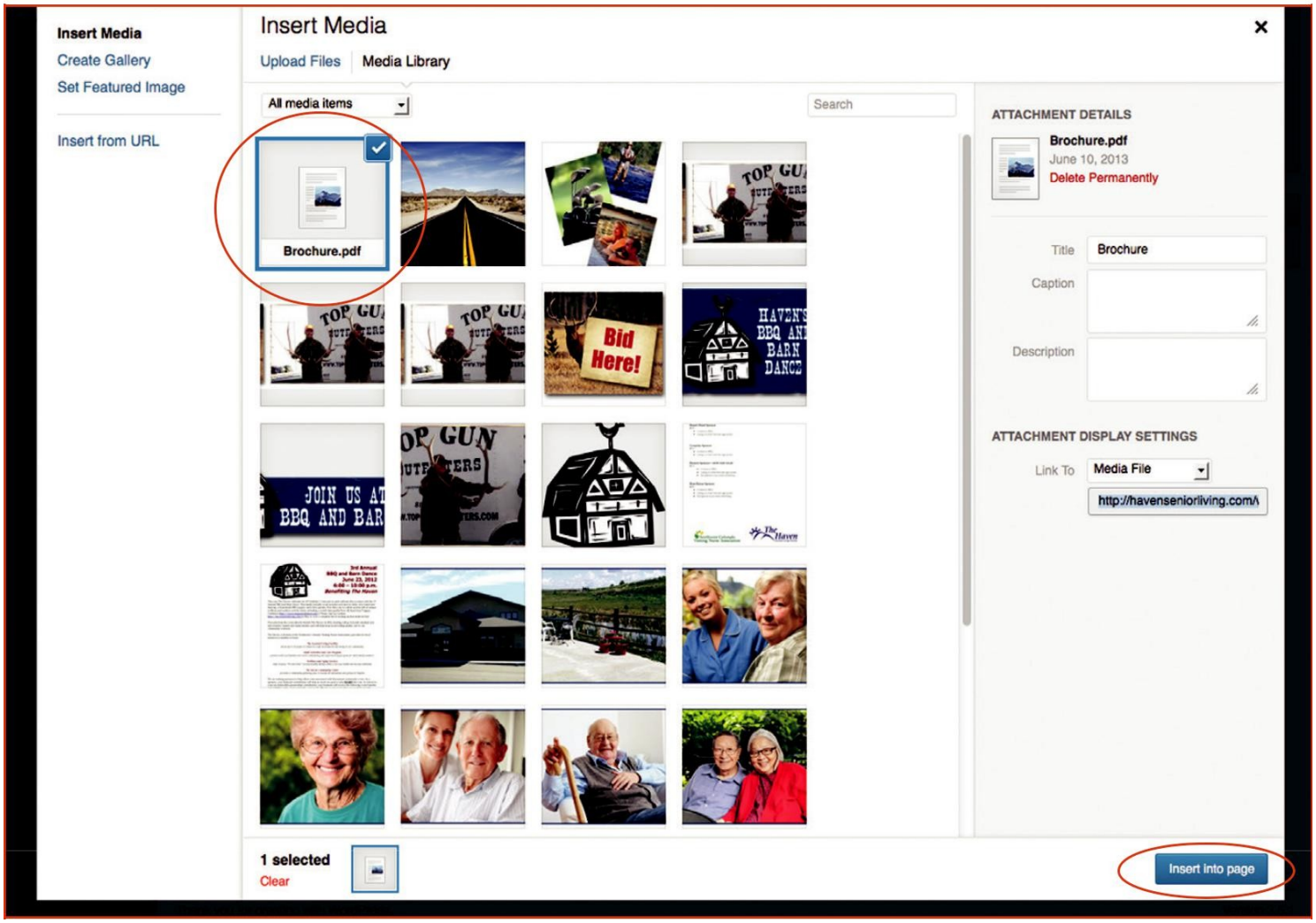




d. To upload a **new PDF**, click “Select Files”.

1. A new window will pop-up - Select the PDF you would like to add - Click “Open”.





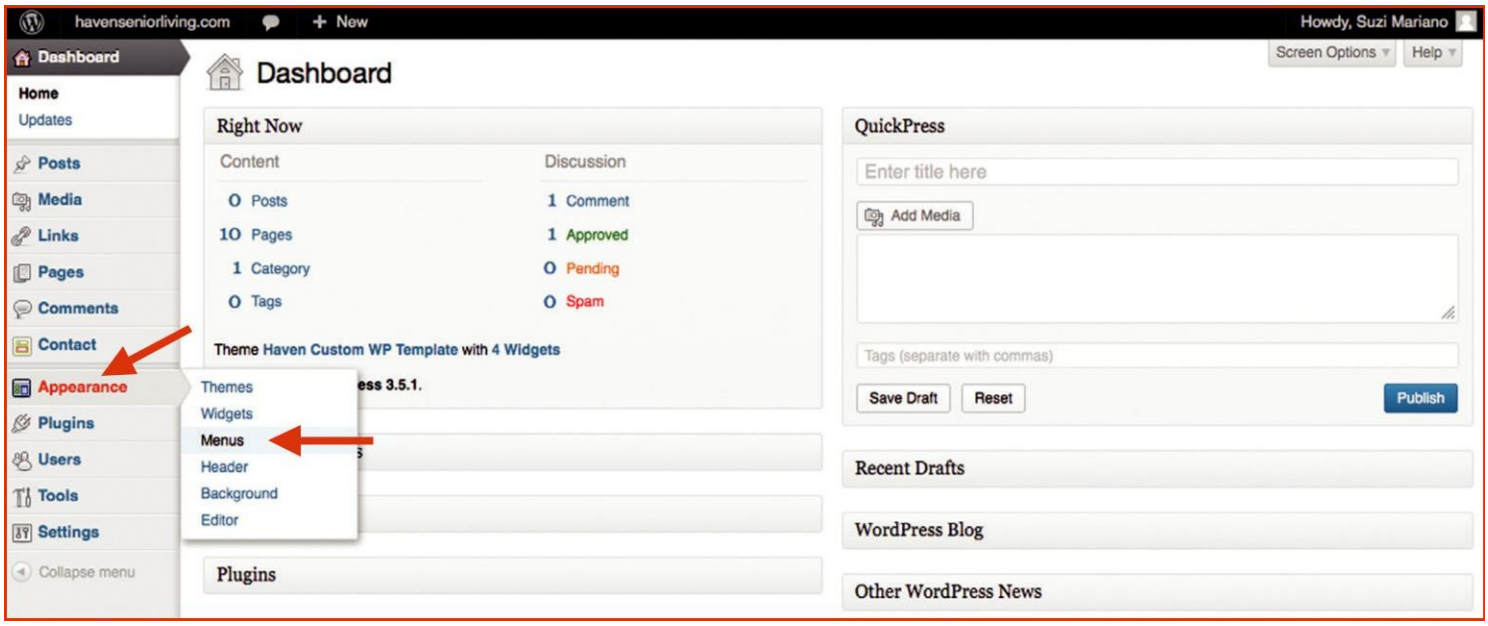
3. Your PDF has been added to your web page and the PDF will open when visitors click on the link.
4. Remember to click update to save any additions.

5. To add a pdf you have already uploaded go to the "**Media Library**", click on "Media Library".
    a. Click on the pdf you would like to upload.
    b. It will become highlighted with a check mark.
    c. Click "Insert into Page".

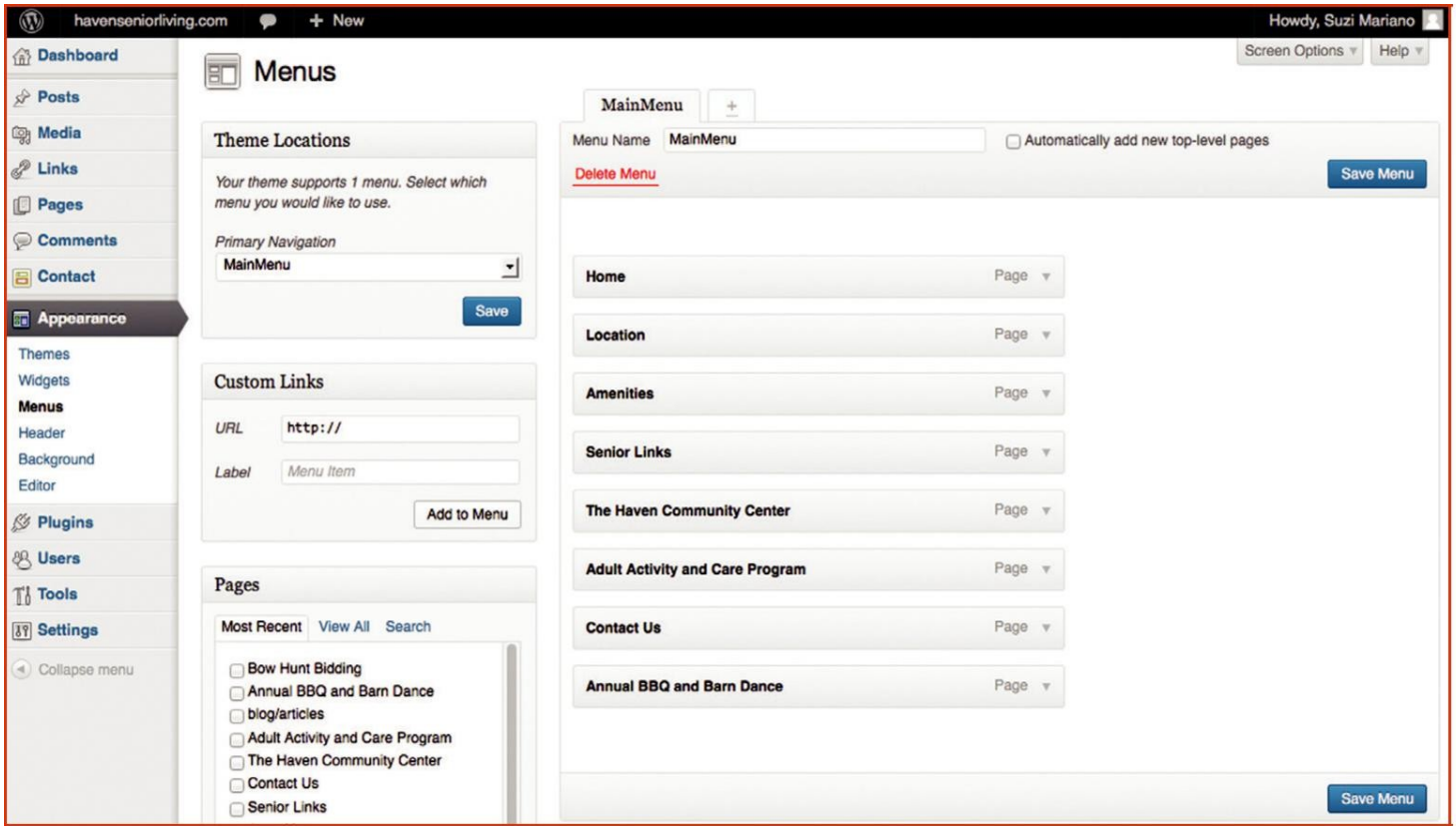
10. Adding Pages to the Menu/Header of My Website

a. From your dashboard:

1. Hover over “Appearance” link.
2. Submenu will appear, click on “Menu”.

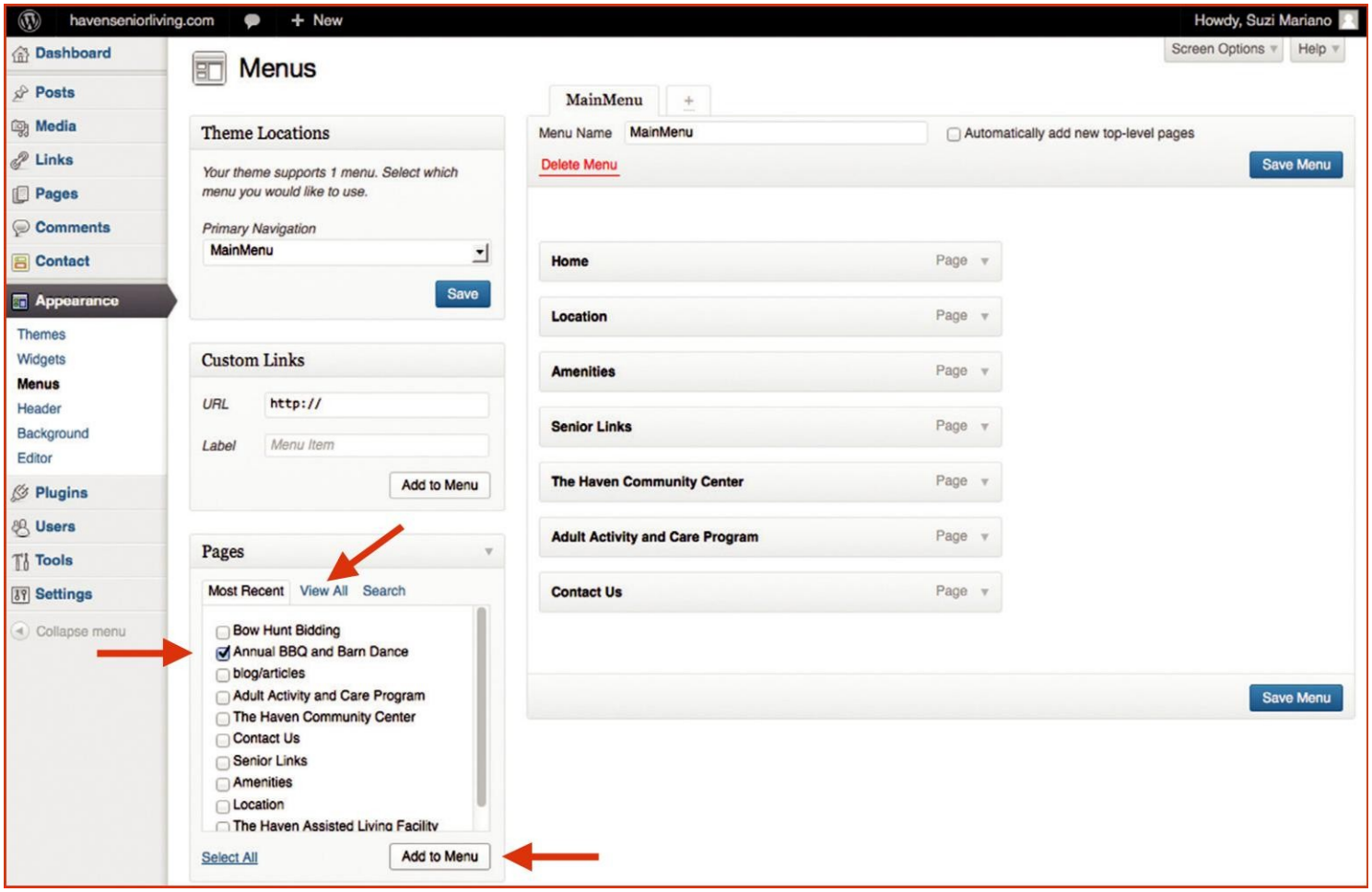


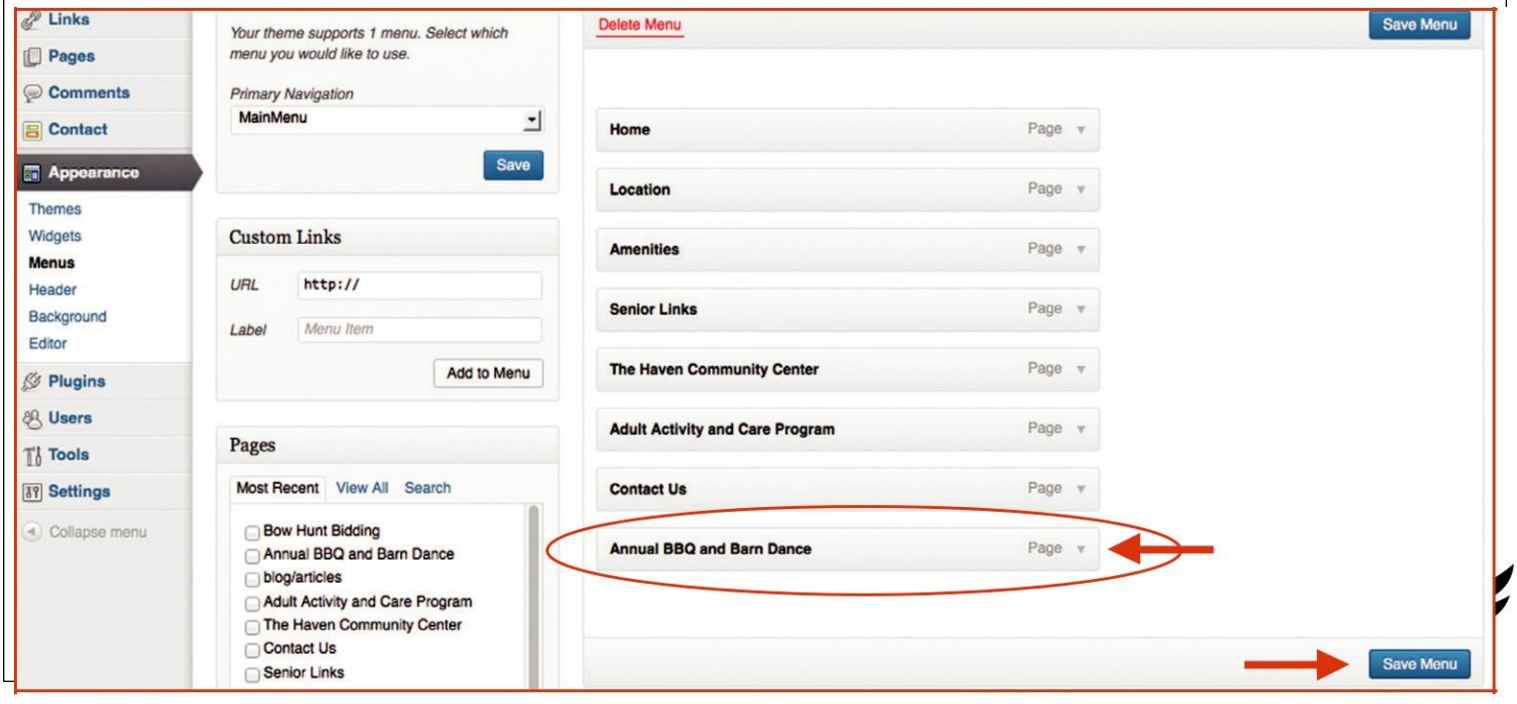
b. Header Menu will pop-up (See below).



10. Adding Pages to the Menu/Header of My Website - Continued

1. Click on page to add.
2. Click “Add to Menu”.
3. If you do not see the page you would like to add, click “View All”. This will list all your pages.

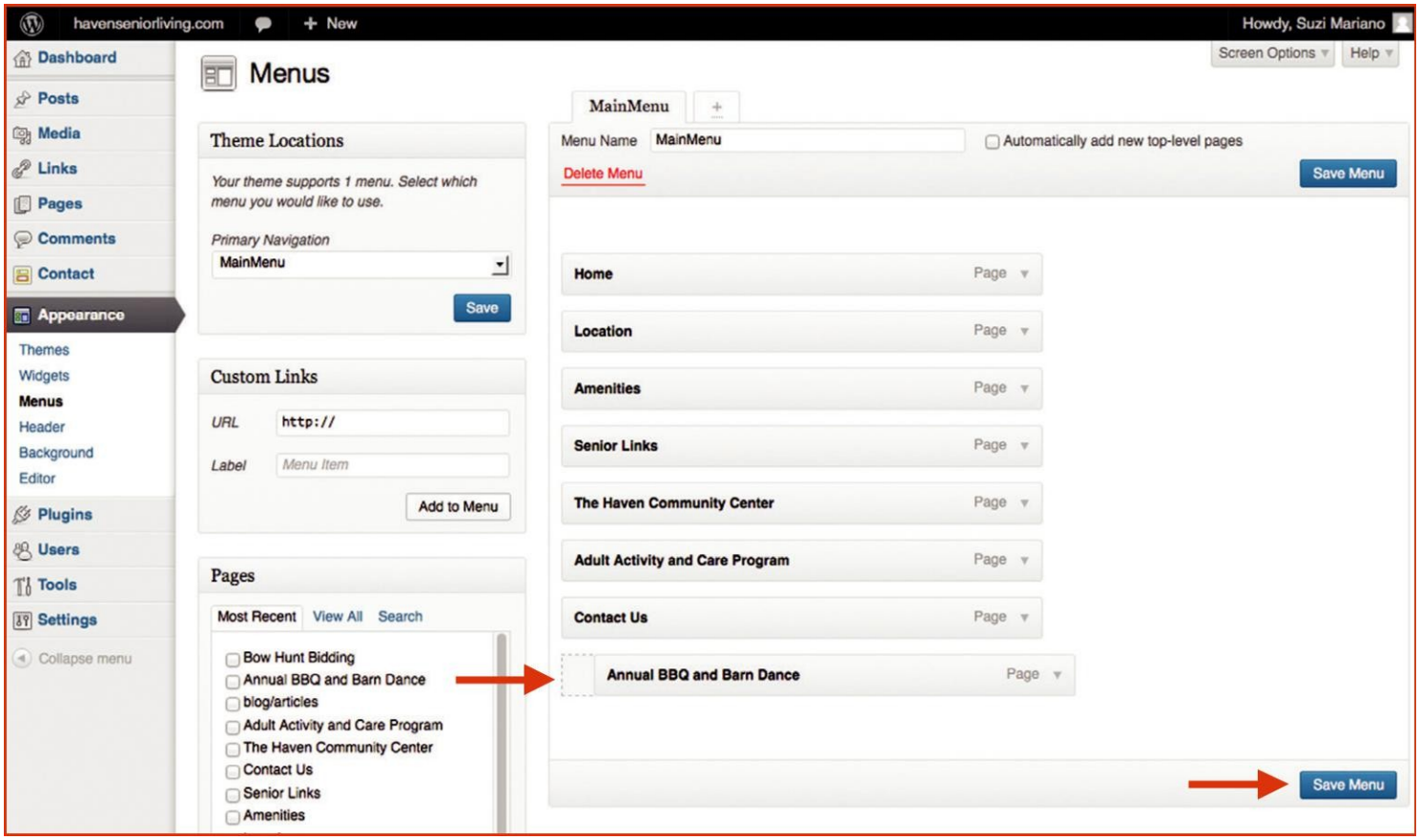




11. Adding Sub-Pages to the Menu/Header of My Website

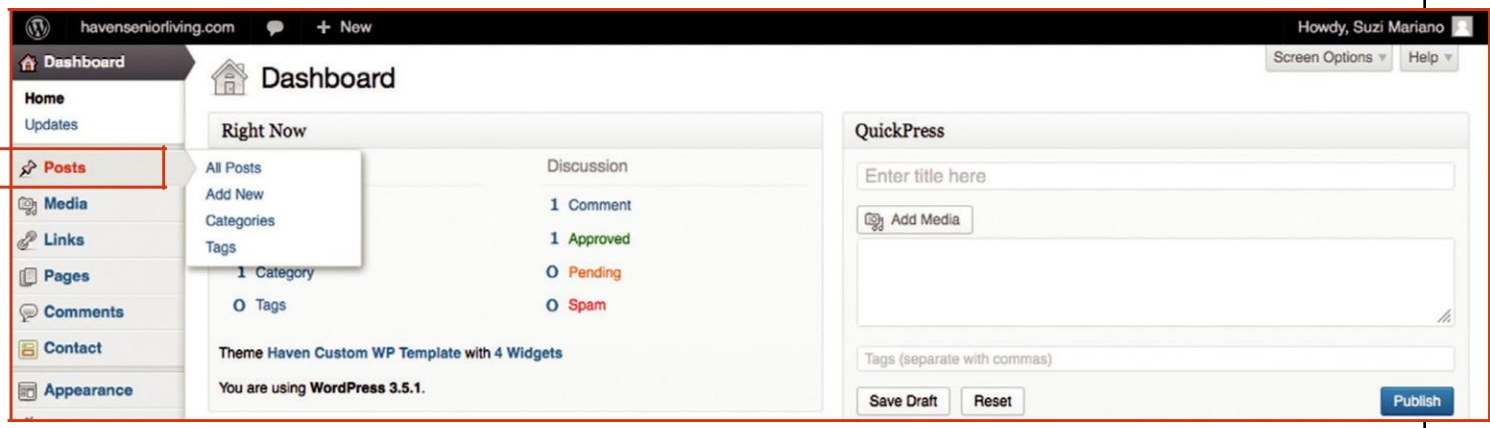
a. Follow earlier steps to get to your menu.

1. Click on menu item, hold down and drag to right of parent menu item.
2. Click “Save Menu”.



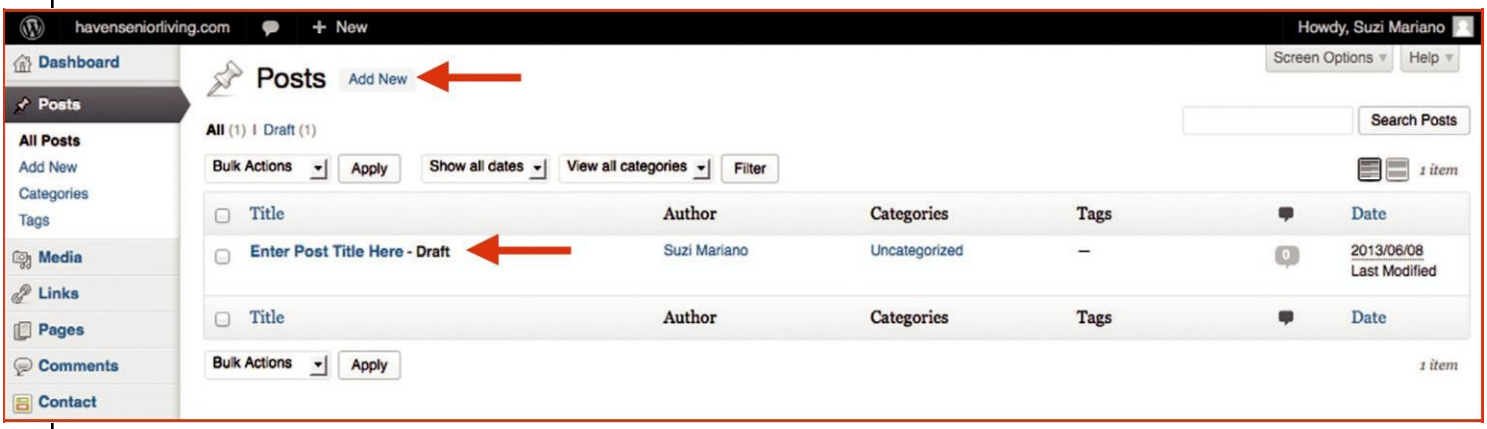
12. **Adding Blog Posts to Your Website**

- From your dashboard, click on "Posts".



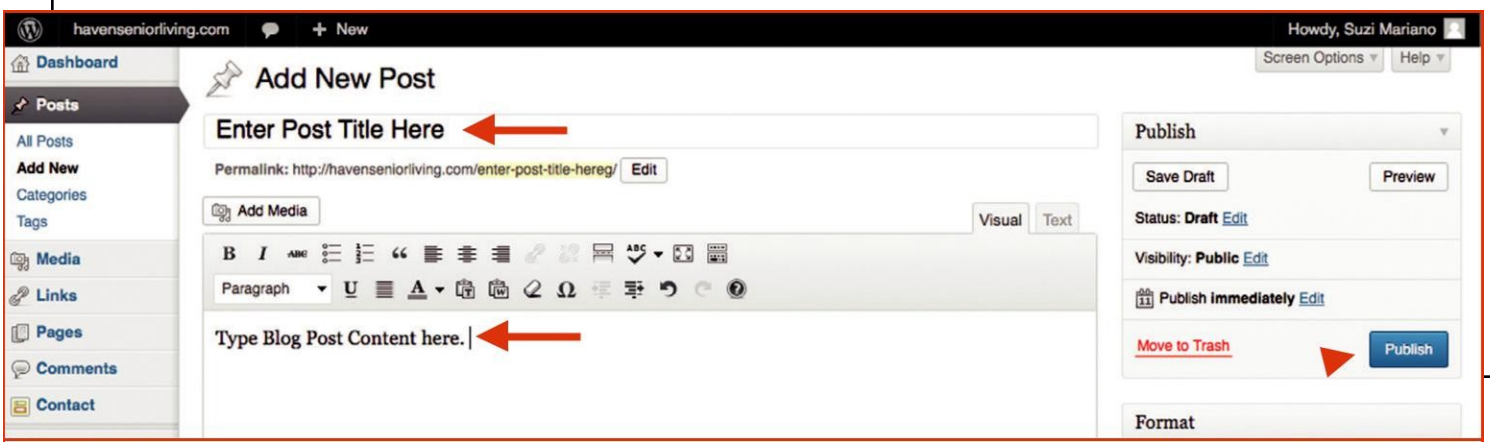
The Posts menu will pop-up. There you can:

- Add a new post. Click "Add New".
- Edit a current post. Click on post in listing.



/ Then your new/current post will pop up. Add your post content.

- Type your blog post title.
- Type your blog post content.
- Click "Publish" to save your blog post.



OOP

using

PHP

What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Classes and Objects

Classes and objects are the two main aspects of object-oriented programming.

A class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

```
<?php
   //OOP-OBJECT ORIENTED PROGRAMMING
   class Animal
   {

      var $name;
```

```
    function set_name($text){

        $this->name = $text;

    }

    function get_name()
    {

                //echo "The new lion is ",$this->name , ".";

        return $this->name;

    }

}

$lion = new Animal();//lion is an object of class animal

$lion->set_name("Leo");

        //$lion->get_name();

echo "The name of your new lion is ".$lion->get_name();
        ?>
```

Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the `new` keyword.

The $this Keyword

The $this keyword refers to the current object, and is only available inside methods

OOP - Constructor

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (__)!

```
<?php
//CONSTRUCTOR
class test
{

        var $sal;

        var $da;

        var $net;

        function __construct($a=0,$b=0)//constructor
        {

                $this->sal=$a;

                $this->da=$b;

        }
        function add()
```

```
        {

                $this->net= $this->sal+$this->da;

                return $this->net;

        }

        function __destruct()
        {

                echo "<br>"."destructor called";

        }

}
  $obj= new test(2000,800);//constructor called

  echo "the  sum is  ". $obj->add();
  ?>
```

The __destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a __destruct() function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (__)!

OOP - Inheritance

What is Inheritance?

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the `extends` keyword.

```
<?php
//INHERITANCE
class Stud
{

	var $name;

	function input($txt)
	{

		$this->name=$txt;

	}
	function output()
	{

		echo "Name: ". $this->name;
```

```
        }

}
class test extends Stud//test is the child class of Stud
{

   var $marks1,$marks2;

   function in($a,$b)
        {

          echo  "within test ";
          $this->marks1=$a;
          $this->marks2=$b;

        }

        function out()
        {

          $this->output();

          echo "<br>The first marks is ".$this->marks1;

          echo "<br>The second marks is ".$this->marks2;

        }

}
```

```
$obj=new test();


$obj->input("test");
$obj->in(60,80);
$obj->out();
?>
```

Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- public - the property or method can be accessed from everywhere. This is default
- protected - the property or method can be accessed within the class and by classes derived from that class
- private - the property or method can ONLY be accessed within the class

Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

```
<?php
//METHOD OVERRIDING-polymorphism(runtime)
class base
{

    function f1()
    {

        echo "<br>in parent class";
```

```
        }


}
class child extends base
{
   function f1()
        {
                parent::f1();//invokes parent class function f1


                echo "<br>in child class";
        }
}
$obj=new child();
$obj->f1();
?>
```

Class Constants

Constants cannot be changed once it is declared.

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the `const` keyword.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (`::`) followed by the constant name, like here:

```
<?php
//CLASS CONSTANTS
class test
{


        const c1="hello world";
```

```
        function f1()
        {

                echo self::c1;

        }
}

$obj=new test();

$obj->f1();

echo "<br>";

echo $obj::c1;

echo "<br>";

echo test::c1;

?>
```

What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the `abstract` keyword

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may has optional arguments in addition

```
abstract class MyAbstractClass {
  abstract function myAbstractFunction() {
  }
}
```

Static Methods

Static methods can be called directly - without creating an instance of a class.

Static methods are declared with the `static` keyword

To access a static method use the class name, double colon (::), and the method name:

Syntax

ClassName::staticMethod();

A class can have both static and non-static methods. A static method can be accessed from a method in the same class using the self keyword and double colon (::).

Interfaces

Interfaces are defined to provide a common function names to the implementers. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

```
<?php
interface i1
{

        function area($s,$t);//function declaration
}
class square implements i1

{

        function area($s,$t)
        {

                $s=$t;

                return $s*$s;

        }

}
class rect implements i1
{

        function area($s,$t)
        {

                return $s*$t;
```

```
    }

}

    $width=4;

    $height=6;

    $obj=new square();

    echo $obj->area($width,$height) . "<br>";

    $obj=new rect();

    echo $obj->area($width,$height). "<br>";
```

Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.

Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php

  class BaseClass {
    public function test() {
      echo "BaseClass::test() called<br>";
    }

    final public function moreTesting() {
      echo "BaseClass::moreTesting() called<br>";
```

```
    }
  }

  class ChildClass extends BaseClass {
    public function moreTesting() {
      echo "ChildClass::moreTesting() called<br>";
    }
  }
?>
```

MVC WITH CODEIGNITER

What is CodeIgniter?

CodeIgniter is an Application Framework

CodeIgniter is a toolkit for people who build web applications using PHP. Its goal is to enable you to develop projects much faster than you could if you were writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. CodeIgniter lets you creatively focus on your project by minimizing the amount of code needed for a given task.

CodeIgniter is Free

CodeIgniter is licensed under the MIT license so you can use it however you please. For more information please read the license agreement.

CodeIgniter is Light Weight

Truly light weight. The core system requires only a few very small libraries. This is in stark contrast to many frameworks that require significantly more resources. Additional libraries are loaded dynamically upon request, based on your needs for a given process, so the base system is very lean and quite fast.

CodeIgniter is Fast

Really fast. We challenge you to find a framework that has better performance than CodeIgniter.

CodeIgniter Uses M-V-C

CodeIgniter uses the Model-View-Controller approach, which allows great separation between logic and presentation. This is particularly good for projects in which designers are working with your template files, as the code these files contain will be minimized. We describe MVC in more detail on its own page.

CodeIgniter Generates Clean URLs

The URLs generated by CodeIgniter are clean and search-engine friendly. Rather than using the standard “query string” approach to URLs that is synonymous with dynamic systems, CodeIgniter uses a segment-based approach:

example.com/news/article/345

CodeIgniter Packs a Punch

CodeIgniter comes with full-range of libraries that enable the most commonly needed web development tasks, like accessing a database, sending email, validating form data, maintaining sessions, manipulating images, working with XML-RPC data and much more.

CodeIgniter is Extensible

The system can be easily extended through the use of your own libraries, helpers, or through class extensions or system hooks.

CodeIgniter Does Not Require a Template Engine

Although CodeIgniter *does* come with a simple template parser that can be optionally used, it does not force you to use one. Template engines simply can not match the performance of native PHP, and the syntax that must be learned to use a template engine is usually only marginally easier than learning the basics of PHP. Consider this block of PHP code:

<?php **foreach** ($addressbook **as** $name)**:?>**

<?=$name?>

<?phpendforeach; **?>**

Contrast this with the pseudo-code used by a template engine:

{**foreach** from=$addressbook item="name"}

{$name}

{/**foreach**}

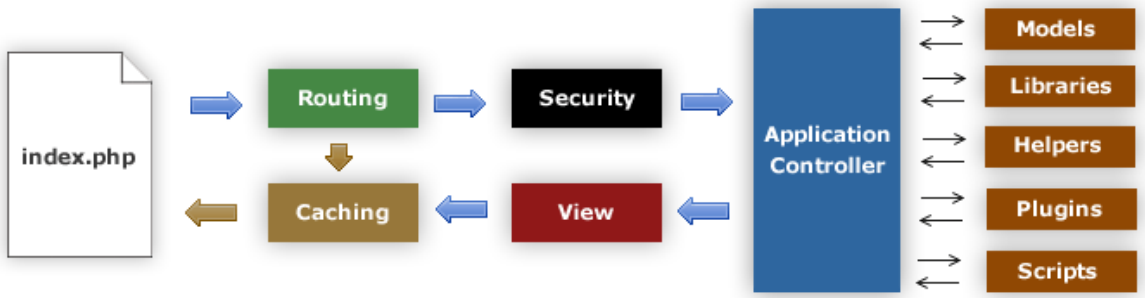
Yes, the template engine example is a bit cleaner, but it comes at the price of performance, as the pseudo-code must be converted back into PHP to run. Since one of our goals is *maximum performance*, we opted to not require the use of a template engine.

CodeIgniter is Thoroughly Documented

Programmers love to code and hate to write documentation. We're no different, of course, but since documentation is **as important** as the code itself, we are committed to doing it. Our source code is extremely clean and well commented as well.

CodeIgniter has a Friendly Community of Users

Application Flow Chart



1. The index.php serves as the front controller, initializing the base resources needed to run CodeIgniter.
2. The Router examines the HTTP request to determine what should be done with it.
3. If a cache file exists, it is sent directly to the browser, bypassing the normal system execution.
4. Security. Before the application controller is loaded, the HTTP request and any user submitted data is filtered for security.
5. The Controller loads the model, core libraries, helpers, and any other resources needed to process the specific request.
6. The finalized View is rendered then sent to the web browser to be seen. If caching is enabled, the view is cached first so that on subsequent requests it can be served.

Model-View-Controller

CodeIgniter is based on the Model-View-Controller development pattern. MVC is a software approach that separates application logic from presentation. In practice, it permits your web pages to contain minimal scripting since the presentation is separate from the PHP scripting.

- The **Model** represents your data structures. Typically your model classes will contain functions that help you retrieve, insert, and update information in your database.

- The **View** is the information that is being presented to a user. A View will normally be a web page, but in CodeIgniter, a view can also be a page fragment like a header or footer. It can also be an RSS page, or any other type of "page".
- The **Controller** serves as an *intermediary* between the Model, the View, and any other resources needed to process the HTTP request and generate a web page.

CodeIgniter has a fairly loose approach to MVC since Models are not required. If you don't need the added separation, or find that maintaining models requires more complexity than you want, you can ignore them and build your application minimally using Controllers and Views. CodeIgniter also enables you to incorporate your own existing scripts, or even develop core libraries for the system, enabling you to work in a way that makes the most sense to you.

Sample Controller:hello.php

```
<?php

    class hello extends CI_Controller
    {
        var $x;
        var $dt;
        public function hello()
        {
            parent::__construct();

            $this->x=6;
            $dt=array();



        }

        function display()
        {
            $this->dt['viewvar']=$this->x;
            $this->load->view('hello',$this->dt);
```

```
        }
    }


?>
```

Sample view file: hello.php

```
<?php

    echo $viewvar;


?>
```

Config Class

The Config class provides a means to retrieve configuration preferences. These preferences can come from the default config file (application/config/config.php) or from your own custom config files.

Anatomy of a Config File

By default, CodeIgniter has one primary config file, located at application/config/config.php. If you open the file using your text editor you'll see that config items are stored in an array called $config.

You can add your own config items to this file, or if you prefer to keep your configuration items separate (assuming you even need config items), simply create your own file and save it in config folder.

CodeIgniter automatically loads the primary config file (application/config/config.php), so you will only need to load a config file if you have created your own.

Loader Class

Loader, as the name suggests, is used to load elements. These elements can be libraries (classes) View files, Drivers, Helpers, Models, or your own files

Input Class

The Input Class serves two purposes:

1. It pre-processes global input data for security.
2. It provides some helper methods for fetching input data and pre-processing it.

Note

This class is initialized automatically by the system so there is no need to do it manually.

Accessing form data

Using POST, GET, COOKIE, or SERVER Data

CodeIgniter comes with helper methods that let you fetch POST, GET, COOKIE or SERVER items. The main advantage of using the provided methods rather than fetching an item directly ($_POST['something']) is that the methods will check to see if the item is set and return NULL if not. This lets you conveniently use data without having to test whether an item exists first. In other words, normally you might do something like this:

$something = isset($_POST['something']) **?** $_POST['something'] **:NULL**;

With CodeIgniter's built in methods you can simply do this:

$something = $this->input->post('something');

The main methods are:

- $this->input->post()
- $this->input->get()
- $this->input->cookie()
- $this->input->server()

Form Validation

CodeIgniter provides a comprehensive form validation and data prepping class that helps minimize the amount of code you'll write.

Before explaining CodeIgniter's approach to data validation, let's describe the ideal scenario:

1. A form is displayed.
2. You fill it in and submit it.

3. If you submitted something invalid, or perhaps missed a required item, the form is redisplayed containing your data along with an error message describing the problem.
4. This process continues until you have submitted a valid form.

On the receiving end, the script must:

1. Check for required data.
2. Verify that the data is of the correct type, and meets the correct criteria. For example, if a username is submitted it must be validated to contain only permitted characters. It must be of a minimum length, and not exceed a maximum length. The username can't be someone else's existing username, or perhaps even a reserved word. Etc.
3. Sanitize the data for security.
4. Pre-format the data if needed (Does the data need to be trimmed? HTML encoded? Etc.)
5. Prep the data for insertion in the database.

Although there is nothing terribly complex about the above process, it usually requires a significant amount of code, and to display error messages, various control structures are usually placed within the form HTML. Form validation, while simple to create, is generally very messy and tedious to implement.

Form Validation Tutorial

What follows is a "hands on" tutorial for implementing CodeIgniter's Form Validation.

In order to implement form validation you'll need three things:

1. A View file containing a form.
2. A View file containing a "success" message to be displayed upon successful submission.
3. A controller method to receive and process the submitted data.

Let's create those three things, using a member sign-up form as the example.

The Form

Using a text editor, create a form called myform.php. In it, place this code and save it to your application/views/ folder:

```
<html>
<head>
<title>My Form</title>
</head>
<body>
```

```
<?php echo validation_errors(); ?>

<?php echo form_open('form'); ?>

<h5>Username</h5>
<input type="text" name="username" value="" size="50" />

<h5>Password</h5>
<input type="text" name="password" value="" size="50" />

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="" size="50" />

<h5>Email Address</h5>
<input type="text" name="email" value="" size="50" />

<div><input type="submit" value="Submit" /></div>

</form>

</body>
</html>
```

The Success Page

Using a text editor, create a form called formsuccess.php. In it, place this code and save it to your application/views/ folder:

```
<html>
<head>
<title>My Form</title>
</head>
<body>
```

```
<h3>Your form was successfully submitted!</h3>

<p><?php echo anchor('form', 'Try it again!'); ?></p>

</body>
</html>
```

The Controller

Using a text editor, create a controller called Form.php. In it, place this code and save it to your application/controllers/ folder:

```
<?php

Class Form extends CI_Controller {

public function index()
    {
        $this->load->helper(array('form', 'url'));

        $this->load->library('form_validation');

if ($this->form_validation->run() ==FALSE)
        {
            $this->load->view('myform');
        }
else
        {
```

```
                $this->load->view('formsuccess');
            }
        }
    }
```

To try your form, visit your site using a URL similar to this one:

Localhost/<codeigniter application folder>index**.**php/form/

If you submit the form you should simply see the form reload. That's because you haven't set up any validation rules yet.

Since you haven't told the Form Validation class to validate anything yet, it returns FALSE (boolean false) by default. ``The run()`` method only returns TRUE if it has successfully applied your rules without any of them failing.

Explanation

You'll notice several things about the above pages:

The form (myform.php) is a standard web form with a couple exceptions:

1. It uses a form helper to create the form opening. Technically, this isn't necessary. You could create the form using standard HTML. However, the benefit of using the helper is that it generates the action URL for you, based on the URL in your config file. This makes your application more portable in the event your URLs change.
2. At the top of the form you'll notice the following function call:
3. **<?**php **echo** validation_errors(); **?>**

This function will return any error messages sent back by the validator. If there are no messages it returns an empty string.

The controller (Form.php) has one method: index(). This method initializes the validation class and loads the form helper and URL helper used by your view files. It also runs the validation routine. Based on whether the validation was successful it either presents the form or the success page.

Setting Validation Rules

CodeIgniter lets you set as many validation rules as you need for a given field, cascading them in order, and it even lets you prep and pre-process the field data at the same time. To set validation rules you will use the set_rules() method:

$this->form_validation->set_rules();

The above method takes **three** parameters as input:

1. The field name - the exact name you've given the form field.

2. A “human” name for this field, which will be inserted into the error message. For example, if your field is named “user” you might give it a human name of “Username”.
3. The validation rules for this form field.
4. (optional) Set custom error messages on any rules given for current field. If not provided will use the default one.

Note

If you would like the field name to be stored in a language file, please see Translating Field Names.

Here is an example. In your controller (Form.php), add this code just below the validation initialization method:

```
$this->form_validation->set_rules('username', 'Username', 'required');

$this->form_validation->set_rules('password', 'Password', 'required');

$this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');

$this->form_validation->set_rules('email', 'Email', 'required');
```

Your controller should now look like this:

```
<?php

Class Form extends CI_Controller {

public function index()
    {
        $this->load->helper(array('form', 'url'));

        $this->load->library('form_validation');

        $this->form_validation->set_rules('username', 'Username', 'required');
        $this->form_validation->set_rules('password', 'Password', 'required',
array('required' => 'You must provide a %s.')
        );
        $this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');
        $this->form_validation->set_rules('email', 'Email', 'required');
```

```
if ($this->form_validation->run() ==FALSE)
        {
            $this->load->view('myform');
        }
    else
        {
            $this->load->view('formsuccess');
        }
    }
}
```

Now submit the form with the fields blank and you should see the error messages. If you submit the form with all the fields populated you'll see your success page.

Note

The form fields are not yet being re-populated with the data when there is an error. We'll get to that shortly.

Setting Rules Using an Array

Before moving on it should be noted that the rule setting method can be passed an array if you prefer to set all your rules in one action. If you use this approach, you must name your array keys as indicated:

```
$config =array(
    array(
            'field' => 'username',
            'label' => 'Username',
            'rules' => 'required'
        ),
    array(
            'field' => 'password',
            'label' => 'Password',
            'rules' => 'required',
            'errors' =>array(
                'required' => 'You must provide a %s.',
            ),
```

```
        ),
    array(
            'field' => 'passconf',
            'label' => 'Password Confirmation',
            'rules' => 'required'
        ),
    array(
            'field' => 'email',
            'label' => 'Email',
            'rules' => 'required'
        )
    );

    $this->form_validation->set_rules($config);
```

Cascading Rules

CodeIgniter lets you pipe multiple rules together. Let's try it. Change your rules in the third parameter of rule setting method, like this:

```
    $this->form_validation->set_rules(
        'username', 'Username',
        'required|min_length[5]|max_length[12]|is_unique[users.username]',
    array(
            'required'      => 'You have not provided %s.',
            'is_unique'     => 'This %s already exists.'
        )
    );
    $this->form_validation->set_rules('password', 'Password', 'required');
    $this->form_validation->set_rules('passconf', 'Password Confirmation', 'required|
matches[password]');
    $this->form_validation->set_rules('email', 'Email', 'required|valid_email|is_unique[users.email]');
```

The above code sets the following rules:

1. The username field be no shorter than 5 characters and no longer than 12.
2. The password field must match the password confirmation field.
3. The email field must contain a valid email address.

Give it a try! Submit your form without the proper data and you'll see new error messages that correspond to your new rules. There are numerous rules available which you can read about in the validation reference.

Note

You can also pass an array of rules to set_rules(), instead of a string. Example:

$this->form_validation->set_rules('username', 'Username', **array**('required', 'min_length[5]'));

Prepping Data

In addition to the validation method like the ones we used above, you can also prep your data in various ways. For example, you can set up rules like this:

$this->form_validation->set_rules('username', 'Username', 'trim|required|min_length[5]| max_length[12]');

$this->form_validation->set_rules('password', 'Password', 'trim|required|min_length[8]');

$this->form_validation->set_rules('passconf', 'Password Confirmation', 'trim|required| matches[password]');

$this->form_validation->set_rules('email', 'Email', 'trim|required|valid_email');

In the above example, we are "trimming" the fields, checking for length where necessary and making sure that both password fields match.

Any native PHP function that accepts one parameter can be used as a rule, like ``htmlspecialchars()``, ``trim()``, etc.

Note

You will generally want to use the prepping functions **after** the validation rules so if there is an error, the original data will be shown in the form.

Re-populating the form

Thus far we have only been dealing with errors. It's time to repopulate the form field with the submitted data. CodeIgniter offers several helper functions that permit you to do this. The one you will use most commonly is:

set_value('field name')

Open your myform.php view file and update the **value** in each field using the set value() function:

Don’t forget to include each field name in the :php:func:`set_value()` function calls!

```
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>

<?php echo form_open('form'); ?>

<h5>Username</h5>
<input type="text" name="username" value="<?php echo set_value('username'); ?>" size="50" />

<h5>Password</h5>
<input type="text" name="password" value="<?php echo set_value('password'); ?>" size="50" />

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="<?php echo set_value('passconf'); ?>" size="50" />

<h5>Email Address</h5>
<input type="text" name="email" value="<?php echo set_value('email'); ?>" size="50" />

<div><input type="submit" value="Submit" /></div>

</form>

</body>
</html>
```

Now reload your page and submit the form so that it triggers an error. Your form fields should now be re-populated

File Uploading Class

CodeIgniter's File Uploading Class permits files to be uploaded. You can set various preferences, restricting the type and size of the files

The Process

Uploading a file involves the following general process:

- An upload form is displayed, allowing a user to select a file and upload it.
- When the form is submitted, the file is uploaded to the destination you specify.
- Along the way, the file is validated to make sure it is allowed to be uploaded based on the preferences you set.
- Once uploaded, the user will be shown a success message.

To demonstrate this process here is brief tutorial. Afterward you'll find reference information.

Creating the Upload Form

Using a text editor, create a form called upload_form.php. In it, place this code and save it to your **application/views/** directory:

```
<html>

<head>

<title>Upload Form</title>

</head>

<body>


<?php echo $error;?>


<?php echo form_open_multipart('upload/do_upload');?>


<input type="file" name="userfile" size="20" />


<br /><br />


<input type="submit" value="upload" />
```

```
</form>


</body>

</html>
```

You'll notice we are using a form helper to create the opening form tag. File uploads require a multipart form, so the helper creates the proper syntax for you. You'll also notice we have an $error variable. This is so we can show error messages in the event the user does something wrong.

The Success Page

Using a text editor, create a form called upload_success.php. In it, place this code and save it to your **application/views/** directory:

```
<html>

<head>

<title>Upload Form</title>

</head>

<body>


<h3>Your file was successfully uploaded!</h3>


<ul>

<?php foreach ($upload_data as $item => $value):?>

<li><?php echo $item;?>: <?phpecho $value;?></li>

<?php endforeach; ?>

</ul>


<p><?php echo anchor('upload', 'Upload Another File!'); ?></p>


</body>

</html>
```

The Controller

Using a text editor, create a controller called Upload.php. In it, place this code and save it to your **application/controllers/** directory:

```
<?php
```

```
Class Upload extends CI_Controller {

public function __construct()
    {
parent::__construct();
        $this->load->helper(array('form', 'url'));
    }

Public function index()
    {
        $this->load->view('upload_form', array('error' => ' ' ));
    }

Public function do_upload()
    {
        $config['upload_path']          = './uploads/';
        $config['allowed_types']        = 'gif|jpg|png';
        $config['max_size']             = 100;
        $config['max_width']            = 1024;
        $config['max_height']           = 768;

        $this->load->library('upload', $config);

if ( ! $this->upload->do_upload('userfile'))
        {
            $error =array('error' => $this->upload->display_errors());

            $this->load->view('upload_form', $error);
        }
else
```

```
            {
                $data =array('upload_data' => $this->upload->data());


                $this->load->view('upload_success', $data);
            }
        }
}
?>
```

The Upload Directory

You'll need a destination directory for your uploaded images. Create a directory at the root of your CodeIgniter installation called uploads and set its file permissions to 777.

Try it!

To try your form, visit your site using a URL similar to this one:

Localhost<folder name>/index.php/upload/

You should see an upload form. Try uploading an image file (either a jpg, gif, or png). If the path in your controller is correct it should work

Initializing the Upload Class

Like most other classes in CodeIgniter, the Upload class is initialized in your controller using the $this->load->library() method:

$this->load->library('upload');

Once the Upload class is loaded, the object will be available using: $this->upload

Setting Preferences

Similar to other libraries, you'll control what is allowed to be upload based on your preferences. In the controller you built above you set the following preferences:

$config['upload_path'] = './uploads/';

$config['allowed_types'] = 'gif|jpg|png';

$config['max_size']   = '100';

$config['max_width'] = '1024';

$config['max_height'] = '768';

$this->load->library('upload', $config);

// Alternately you can set preferences by calling the ``initialize()`` method. Useful if you auto-load the class:

$this->upload->initialize($config);

Database

CodeIgniter comes with a full-featured and very fast abstracted database class that supports both traditional structures and Query Builder patterns. The database functions offer clear, simple syntax.

Database Configuration

CodeIgniter has a config file that lets you store your database connection values (username, password, database name, etc.). The config file is located at application/config/database.php. You can also set database connection values for specific environments by placing **database.php** in the respective environment config folder.

The config settings are stored in a multi-dimensional array with this prototype:

$db['default'] =**array**(

'dsn' => '',

'hostname' => 'localhost',

'username' => 'root',

'password' => '',

'database' => 'database_name',

'dbdriver' => 'mysqli',

'dbprefix' => '',

'pconnect' =>**TRUE**,

'db_debug' =>**TRUE**,

'cache_on' =>**FALSE**,

'cachedir' => '',

'char_set' => 'utf8',

'dbcollat' => 'utf8_general_ci',

'swap_pre' => '',

'encrypt' =>**FALSE**,

```
    'compress' =>FALSE,
    'stricton' =>FALSE,
    'failover' =>array()
);
```

Connecting to your Database

There are two ways to connect to a database:

Automatically Connecting

The “auto connect” feature will load and instantiate the database class with every page load. To enable “auto connecting”, add the word database to the library array, as indicated in the following file:

application/config/autoload.php

Manually Connecting

If only some of your pages require database connectivity you can manually connect to your database by adding this line of code in any function where it is needed, or in your class constructor to make the database available globally in that class.

```
$this->load->database();
```

If the above function does **not** contain any information in the first parameter it will connect to the group specified in your database config file. For most people, this is the preferred method of use.

Available Parameters

1. The database connection values, passed either as an array or a DSN string.
2. TRUE/FALSE (boolean). Whether to return the connection ID (see Connecting to Multiple Databases below).
3. TRUE/FALSE (boolean). Whether to enable the Query Builder class. Set to TRUE by default.

Query Basics

Regular Queries

To submit a query, use the query function:

```
$this->db->query('YOUR QUERY HERE');
```

The query() function returns a database result object when “read” type queries are run, which you can use to show your results. When “write” type queries are run it simply returns TRUE or FALSE

depending on success or failure. When retrieving data you will typically assign the query to your own variable, like this:

```
$query = $this->db->query('YOUR QUERY HERE');
```

Simplified Queries

The simple_query method is a simplified version of the $this->db->query() method. It DOES NOT return a database result set, nor does it set the query timer, or compile bind data, or store your query for debugging. It simply lets you submit a query. Most users will rarely use this function.

It returns whatever the database drivers' "execute" function returns. That typically is TRUE/FALSE on success or failure for write type queries such as INSERT, DELETE or UPDATE statements (which is what it really should be used for) and a resource/object on success for queries with fetchable results.

```
if ($this->db->simple_query('YOUR QUERY'))
{
    echo "Success!";
}
else
{
    echo "Query failed!";
}
```

Generating Query Results

There are several ways to generate query results:

- Result Arrays
- Result Rows
- Custom Result Objects
- Result Helper Methods
- Class Reference

Result Arrays

result()

This method returns the query result as an array of **objects**, or **an empty array** on failure. Typically you'll use this in a foreach loop, like this:

```
$query = $this->db->query("YOUR QUERY");


foreach ($query->result() as $row)

{

echo $row->title;

echo $row->name;

echo $row->body;

}
```

The above method is an alias of result_object().

You can also pass a string to result() which represents a class to instantiate for each result object (note: this class must be loaded)

```
$query = $this->db->query("SELECT * FROM users;");


foreach ($query->result('User') as $user)

{

echo $user->name; // access attributes

echo $user->reverse_name(); // or methods defined on the 'User' class

}
```

result_array()

This method returns the query result as a pure array, or an empty array when no result is produced. Typically you'll use this in a foreach loop, like this:

```
$query = $this->db->query("YOUR QUERY");


foreach ($query->result_array() as $row)

{

echo $row['title'];

echo $row['name'];

echo $row['body'];

}
```

Result Rows

row()

This method returns a single result row. If your query has more than one row, it returns only the first row. The result is returned as an **object**. Here's a usage example:

```
$query = $this->db->query("YOUR QUERY");


$row = $query->row();


if (isset($row))
{
echo $row->title;
echo $row->name;
echo $row->body;
}
```

If you want a specific row returned you can submit the row number as a digit in the first parameter:

```
$row = $query->row(5);
```

You can also add a second String parameter, which is the name of a class to instantiate the row with:

```
$query = $this->db->query("SELECT * FROM users LIMIT 1;");
$row = $query->row(0, 'User');


echo $row->name; // access attributes
echo $row->reverse_name(); // or methods defined on the 'User' class
```

row_array()

Identical to the above row() method, except it returns an array. Example:

```
$query = $this->db->query("YOUR QUERY");


$row = $query->row_array();


if (isset($row))
{
echo $row['title'];
echo $row['name'];
```

echo $row['body'];

}

If you want a specific row returned you can submit the row number as a digit in the first parameter:

$row = $query->row_array(5);

In addition, you can walk forward/backwards/first/last through your results using these variations:

$row = $query->first_row()

$row = $query->last_row()

$row = $query->next_row()

$row = $query->previous_row()

By default they return an object unless you put the word "array" in the parameter:

$row = $query->first_row('array')

$row = $query->last_row('array')

$row = $query->next_row('array')

$row = $query->previous_row('array')

Note

All the methods above will load the whole result into memory (prefetching).
Use unbuffered_row() for processing large result sets.

unbuffered_row()

This method returns a single result row without prefetching the whole result in memory as row() does. If your query has more than one row, it returns the current row and moves the internal data pointer ahead.

$query = $this->db->query("YOUR QUERY");

while ($row = $query->unbuffered_row())

{

echo $row->title;

echo $row->name;

echo $row->body;

}

You can optionally pass 'object' (default) or 'array' in order to specify the returned value's type:

$query->unbuffered_row(); *// object*

$query->unbuffered_row('object'); *// object*

```
$query->unbuffered_row('array');        // associative array
```

Custom Result Objects

You can have the results returned as an instance of a custom class instead of a stdClass or array, as the result() and result_array() methods allow. This requires that the class is already loaded into memory. The object will have all values returned from the database set as properties. If these have been declared and are non-public then you should provide a __set() method to allow them to be set.

Example:

```
Class User {


public $id;

public $email;

public $username;


protected $last_login;


public function last_login($format)

    {

return $this->last_login->format($format);

    }


Public function __set($name, $value)

    {

if ($name === 'last_login')

        {

            $this->last_login = DateTime::createFromFormat('U', $value);

        }

    }


Public function __get($name)

    {

if (isset($this->$name))
```

```
            {
return $this->$name;
            }
      }
}
```

In addition to the two methods listed below, the following methods also can take a class name to return the results as: first_row(), last_row(), next_row(), and previous_row().

custom_result_object()

Returns the entire result set as an array of instances of the class requested. The only parameter is the name of the class to instantiate.

Example:

```
$query = $this->db->query("YOUR QUERY");


$rows = $query->custom_result_object('User');


foreach ($rows as $row)
{
echo $row->id;
echo $row->email;
echo $row->last_login('Y-m-d');
}
```

custom_row_object()

Returns a single row from your query results. The first parameter is the row number of the results. The second parameter is the class name to instantiate.

Example:

```
$query = $this->db->query("YOUR QUERY");


$row = $query->custom_row_object(0, 'User');


if (isset($row))
{
echo $row->email;   // access attributes
```

```
echo $row->last_login('Y-m-d');   // access class methods
}
```

You can also use the row() method in exactly the same way.

Example:

```
$row = $query->custom_row_object(0, 'User');
```

Result Helper Methods

num_rows()

The number of rows returned by the query. Note: In this example, $query is the variable that the query result object is assigned to:

```
$query = $this->db->query('SELECT * FROM my_table');


echo $query->num_rows();
```

Note

Not all database drivers have a native way of getting the total number of rows for a result set. When this is the case, all of the data is prefetched and count() is manually called on the resulting array in order to achieve the same result.

num_fields()

The number of FIELDS (columns) returned by the query. Make sure to call the method using your query result object:

```
$query = $this->db->query('SELECT * FROM my_table');


echo $query->num_fields();
```

free_result()

It frees the memory associated with the result and deletes the result resource ID. Normally PHP frees its memory automatically at the end of script execution. However, if you are running a lot of queries in a particular script you might want to free the result after each query result has been generated in order to cut down on memory consumption.

Example:

```
$query = $this->db->query('SELECT title FROM my_table');


foreach ($query->result() as $row)
{
echo $row->title;
```

```
}


$query->free_result();  // The $query result object will no longer be available


$query2 = $this->db->query('SELECT name FROM some_table');


$row = $query2->row();

echo $row->name;

$query2->free_result(); // The $query2 result object will no longer be available
```

data_seek()

This method sets the internal pointer for the next result row to be fetched. It is only useful in combination with unbuffered_row().

It accepts a positive integer value, which defaults to 0 and returns TRUE on success or FALSE on failure.

```
$query = $this->db->query('SELECT `field_name` FROM `table_name`');

$query->data_seek(5); // Skip the first 5 rows

$row = $query->unbuffered_row();
```

Note

Not all database drivers support this feature and will return FALSE. Most notably - you won't be able to use it with PDO.

Class Reference

***class*CI_DB_result**

result([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of requested results - array, object, or class name

Returns: Array containing the fetched rows

Return type: array

A wrapper for the result_array(), result_object() and custom_result_object() methods.

result_array()

Returns: Array containing the fetched rows

Return type: array

Returns the query results as an array of rows, where each row is itself an associative array.

result_object()

Returns: Array containing the fetched rows

Return type: array

Returns the query results as an array of rows, where each row is an object of type stdClass.

custom_result_object(*$class_name*)

Parameters:
- **$class_name** (*string*) – Class name for the resulting rows

Returns: Array containing the fetched rows

Return type: array

Returns the query results as an array of rows, where each row is an instance of the specified class.

row([*$n = 0*[, *$type = 'object'*]])

Parameters:
- **$n** (*int*) – Index of the query results row to be returned
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: The requested row or NULL if it doesn't exist

Return type: mixed

A wrapper for the row_array(), row_object() and ``custom_row_object() methods.

unbuffered_row([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: Next row from the result set or NULL if it doesn't exist

Return type: mixed

Fetches the next result row and returns it in the requested form.

row_array([*$n = 0*])

Parameters:
- **$n** (*int*) – Index of the query results row to be returned

Returns: The requested row or NULL if it doesn't exist

Return type: array

Returns the requested result row as an associative array.

row_object([*$n = 0*])

Parameters:
- **$n** (*int*) – Index of the query results row to be returned

Returns: The requested row or NULL if it doesn't exist

Return type: stdClass

Returns the requested result row as an object of type stdClass.

custom_row_object(*$n*, *$type*)

Parameters:
- **$n** (*int*) – Index of the results row to return
- **$class_name** (*string*) – Class name for the resulting row

Returns: The requested row or NULL if it doesn't exist

Return type: $type

Returns the requested result row as an instance of the requested class.

data_seek([*$n = 0*])

Parameters:
- **$n** (*int*) – Index of the results row to be returned next

Returns: TRUE on success, FALSE on failure

Return type: bool

Moves the internal results row pointer to the desired offset.

set_row(*$key*[, *$value = NULL*])

Parameters:
- **$key** (*mixed*) – Column name or array of key/value pairs
- **$value** (*mixed*) – Value to assign to the column, $key is a single field name

Return type: void

Assigns a value to a particular column.

next_row([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: Next row of result set, or NULL if it doesn't exist

Return type: mixed

Returns the next row from the result set.

previous_row([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: Previous row of result set, or NULL if it doesn't exist

Return type: mixed

Returns the previous row from the result set.

first_row([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: First row of result set, or NULL if it doesn't exist

Return type: mixed

Returns the first row from the result set.

last_row([*$type = 'object'*])

Parameters:
- **$type** (*string*) – Type of the requested result - array, object, or class name

Returns: Last row of result set, or NULL if it doesn't exist

Return type: mixed

Returns the last row from the result set.

num_rows()

Returns: Number of rows in the result set

Return type: int

Returns the number of rows in the result set.

num_fields()

Returns: Number of fields in the result set

Return type: int

Returns the number of fields in the result set.

field_data()

Returns: Array containing field meta-data

Return type: array

Generates an array of stdClass objects containing field meta-data.

free_result()

Return type: void

Frees a result set.

.

list_fields()

Returns: Array of column names

Return type: array

Returns an array containing the field names in the result set.

Query Helper Methods

Information From Executing a Query

$this->db->insert_id()

The insert ID number when performing database inserts.

Note

If using the PDO driver with PostgreSQL, or using the Interbase driver, this function requires a $name parameter, which specifies the appropriate sequence to check for the insert id.

$this->db->affected_rows()

Displays the number of affected rows, when doing “write” type queries (insert, update, etc.).

$this->db->last_query()

Returns the last query that was run (the query string, not the result). Example:

$str = $this->db->last_query();

*// Produces: SELECT * FROM sometable....*

Information About Your Database

$this->db->count_all()

Permits you to determine the number of rows in a particular table. Submit the table name in the first parameter. Example:

echo $this->db->count_all('my_table');

// Produces an integer, like 25

$this->db->platform()

Outputs the database platform you are running (MySQL, MS SQL, Postgres, etc…):

echo $this->db->platform();

$this->db->version()

Outputs the database version you are running:

echo $this->db->version();

Making Your Queries Easier

$this->db->insert_string()

This function simplifies the process of writing database inserts. It returns a correctly formatted SQL insert string. Example:

$data =**array**('name' => $name, 'email' => $email, 'url' => $url);

$str = $this->db->insert_string('table_name', $data);

The first parameter is the table name, the second is an associative array with the data to be inserted. The above example produces:

INSERT INTO table_name (name, email, url) VALUES ('Rick', 'rick@example.com', 'example.com')

Note

Values are automatically escaped, producing safer queries.

$this->db->update_string()

This function simplifies the process of writing database updates. It returns a correctly formatted SQL update string. Example:

```
$data =array('name' => $name, 'email' => $email, 'url' => $url);


$where = "author_id = 1 AND status = 'active'";


$str = $this->db->update_string('table_name', $data, $where);
```

The first parameter is the table name, the second is an associative array with the data to be updated, and the third parameter is the "where" clause. The above example produces:

```
UPDATE table_name SET name = 'Rick', email = 'rick@example.com', url = 'example.com'
WHERE author_id = 1 AND status = 'active'
```

Note

Values are automatically escaped, producing safer queries.

Selecting Data

The following functions allow you to build SQL **SELECT** statements.

$this->db->get()

Runs the selection query and returns the result. Can be used by itself to retrieve all records from a table:

```
$query = $this->db->get('mytable');  // Produces: SELECT * FROM mytable
```

The second and third parameters enable you to set a limit and offset clause:

```
$query = $this->db->get('mytable', 10, 20);


// Executes: SELECT * FROM mytable LIMIT 20, 10

// (in MySQL. Other databases have slightly different syntax)
```

You'll notice that the above function is assigned to a variable named $query, which can be used to show the results:

```
$query = $this->db->get('mytable');


foreach ($query->result() as $row)

{

echo $row->title;
```

}

Selecting Data

The following functions allow you to build SQL **SELECT** statements.

$this->db->get()

Runs the selection query and returns the result. Can be used by itself to retrieve all records from a table:

$query = $this->db->get('mytable'); *// Produces: SELECT * FROM mytable*

The second and third parameters enable you to set a limit and offset clause:

$query = $this->db->get('mytable', 10, 20);

*// Executes: SELECT * FROM mytable LIMIT 20, 10*

// (in MySQL. Other databases have slightly different syntax)

You'll notice that the above function is assigned to a variable named $query, which can be used to show the results:

$query = $this->db->get('mytable');

foreach ($query->result() **as** $row)

{

echo $row->title;

}

Updating Data

$this->db->replace()

This method executes a REPLACE statement, which is basically the SQL standard for (optional) DELETE + INSERT, using *PRIMARY* and *UNIQUE* keys as the determining factor. In our case, it will save you from the need to implement complex logics with different combinations of select(), update(), delete() and insert() calls.

Example:

$data =**array**(

'title' => 'My title',

'name' => 'My Name',

'date' => 'My date'

);

```
$this->db->replace('table', $data);


// Executes: REPLACE INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')
```

In the above example, if we assume that the *title* field is our primary key, then if a row containing 'My title' as the *title* value, that row will be deleted with our new row data replacing it.

Usage of the set() method is also allowed and all fields are automatically escaped, just like with insert().

$this->db->set()

This function enables you to set values for inserts or updates.

It can be used instead of passing a data array directly to the insert or update functions:

```
$this->db->set('name', $name);

$this->db->insert('mytable');  // Produces: INSERT INTO mytable (`name`) VALUES ('{$name}')
```

If you use multiple function called they will be assembled properly based on whether you are doing an insert or an update:

```
$this->db->set('name', $name);

$this->db->set('title', $title);

$this->db->set('status', $status);

$this->db->insert('mytable');
```

set() will also accept an optional third parameter ($escape), that will prevent data from being escaped if set to FALSE. To illustrate the difference, here is set() used both with and without the escape parameter.

```
$this->db->set('field', 'field+1', FALSE);

$this->db->where('id', 2);

$this->db->update('mytable'); // gives UPDATE mytable SET field = field+1 WHERE id = 2


$this->db->set('field', 'field+1');

$this->db->where('id', 2);

$this->db->update('mytable'); // gives UPDATE `mytable` SET `field` = 'field+1' WHERE `id` = 2
```

You can also pass an associative array to this function:

```
$array =array(
        'name' => $name,
        'title' => $title,
        'status' => $status
```

```
);


$this->db->set($array);
$this->db->insert('mytable');
```

Or an object:

```
/*
class Myclass {
        public $title = 'My Title';
        public $content = 'My Content';
        public $date = 'My Date';
}
*/


$object =new Myclass;
$this->db->set($object);
$this->db->insert('mytable');
```

$this->db->update()

Generates an update string and runs the query based on the data you supply. You can pass an **array** or an **object** to the function. Here is an example using an array:

```
$data =array(
        'title' => $title,
        'name' => $name,
        'date' => $date
);


$this->db->where('id', $id);
$this->db->update('mytable', $data);
// Produces:
//
//      UPDATE mytable
//      SET title = '{$title}', name = '{$name}', date = '{$date}'
```

```
//     WHERE id = $id
```

Or you can supply an object:

```
/*
class Myclass {
        public $title = 'My Title';
        public $content = 'My Content';
        public $date = 'My Date';
}
*/


$object =new Myclass;
$this->db->where('id', $id);
$this->db->update('mytable', $object);
// Produces:
//
// UPDATE `mytable`
// SET `title` = '{$title}', `name` = '{$name}', `date` = '{$date}'
// WHERE id = `$id`
```

Note

All values are escaped automatically producing safer queries.

You'll notice the use of the $this->db->where() function, enabling you to set the WHERE clause. You can optionally pass this information directly into the update function as a string:

```
$this->db->update('mytable', $data, "id = 4");
```

Or as an array:

```
$this->db->update('mytable', $data, array('id' => $id));
```

You may also use the $this->db->set() function described above when performing updates.

Deleting Data

$this->db->delete()

Generates a delete SQL string and runs the query.

```
$this->db->delete('mytable', array('id' => $id));  // Produces: // DELETE FROM mytable  // WHERE id = $id
```

The first parameter is the table name, the second is the where clause. You can also use the where() or or_where() functions instead of passing the data to the second parameter of the function:

```
$this->db->where('id', $id);

$this->db->delete('mytable');


// Produces:

// DELETE FROM mytable

// WHERE id = $id
```

An array of table names can be passed into delete() if you would like to delete data from more than 1 table.

```
$tables =array('table1', 'table2', 'table3');

$this->db->where('id', '5');

$this->db->delete($tables);
```

If you want to delete all data from a table, you can use the truncate() function, or empty_table().

$this->db->empty_table()

Generates a delete SQL string and runs the query.:

```
$this->db->empty_table('mytable'); // Produces: DELETE FROM mytable
```

$this->db->truncate()

Generates a truncate SQL string and runs the query.

```
$this->db->from('mytable');

$this->db->truncate();


// or


$this->db->truncate('mytable');


// Produce:

// TRUNCATE mytable
```

Note

If the TRUNCATE command isn't available, truncate() will execute as "DELETE FROM table".