

# Outputting Data

- In PHP there are two basic ways to get output: echo and print Statements

Echo	Print
1. Don't return anything	1. Print has a return value '1'. It can be used in expressions
2. Echo can take many parameters. marginally faster than print	2. Print can take only one argument.

Print :

```
<?php
$text="Hello All";
print "$text to this class";
echo " " . $text. "to this class" ;
?>
```

Hello All to this class

Hello Allto this class

# include() And require() Functions

- Those are useful functions to import files into a PHP script.
- These functions can be used to import external files into a PHP script.
- Example: Assume that on your Web site you have a standard menu bar at the top of every page, and a standard copyright notice in the bottom. Instead of copying and pasting the header and footer code on each individual page, PHP simply create separate files for the header and footer, and import them at the top and bottom of each script. This also makes a change to the site design easier to implement: instead of manually editing a gazillion files, you simply edit two, and the changes are reflected across your entire site instantaneously.

# PHP Include Files

- Code Reusability
- Easy editable
- You can include the content of a PHP file into another PHP file before the server executes it.
- Including files is very useful when you want to include the same PHP text in multiple pages of a website.
  - The include() Function
  - The require() Function

## **PHP include and require Statements:**

- It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the **include or require statement**.

# PHP Include Files Cont....

- The include and require statements are identical, except upon failure:
  - **require** will produce a fatal error (E\_COMPILE\_ERROR) and stop the script.
  - **include** will only produce a warning (E\_WARNING) and the script will continue.
  - Including files saves a lot of work.
- Syntax

```
include 'filename'; OR  
<?php include("menu.html");?>
```

```
require 'filename'; OR  
<?php require("menu.html");?>
```

# Include Example- I

vars.php

```
<?php
```

```
$color = 'green';
```

```
$fruit = 'apple';
```

```
?>
```

test.php

```
<?php
```

```
echo "A $color $fruit";
```

// A

```
include 'vars.php';
```

```
echo "A $color $fruit";
```

// A green apple

```
?>
```

Example: I

- Footer.php

```
<?php
    echo "<p>Copyright &copy; 1999-" . date("Y") . "
    W3Schools.com</p>";
?>
```

- To include the footer file in a page, use the include statement:

```
<html>
  <body>

    <h1>Welcome to my home page!</h1>
    <p>Some text.</p>
    <p>Some more text.</p>
    <?php include 'footer.php';?>

  </body>
</html>
```

**Welcome to my home page!**

Some text.

Some more text.

Copyright © 1999-2017 W3Schools.co

## Example: 2

Assume we have a standard menu file called "menu.php":

```
<?php
    echo '<a href="/default.asp">Home</a> -
    <a href="/html/default.asp">HTML Tutorial</a> -
    <a href="/css/default.asp">CSS Tutorial</a> -
    <a href="/js/default.asp">JavaScript Tutorial</a> -
    <a href="default.asp">PHP Tutorial</a>';
?>
```

```
<html>
    <body>

        <div class="menu">
            <?php include 'menu.php';?>
        </div>

        <h1>Welcome to my home page!</h1>
        <p>Some text.</p>
        <p>Some more text.</p>

    </body>
</html>
```



# Output of Example 2:

[Home](#) - [HTML Tutorial](#) - [CSS Tutorial](#) - [JavaScript Tutorial](#) - [PHP Tutorial](#)

**Welcome to my home page!**

Some text.

Some more text.



# PHP include vs. require

```
<html>
  <body>
    <h1>Welcome to my home page!
  </h1>
  <?php include 'noFileExists.php';
  echo "I have a $color $car.";
?>
</body>
</html>
```

O/P:

**Welcome to my home page!**

I have a .

```
<html>
  <body>
    <h1>Welcome to my home page!</h1>
    <?php require 'noFileExists.php';
    echo "I have a $color $car.";
  ?>
</body>
</html>
```

O/P:

**Welcome to my home page!**

- Use **require** when the file is required by the application.
- Use **include** when the file is not required and application should continue when file is not found.

# include vs. require

include()	require()
The <b>include()</b> function does not stop the execution of the script even if any error occurs.	The <b>require()</b> function will stop the execution of the script when an error occurs.
The <b>include()</b> function does not give a fatal error.	The <b>require()</b> function gives a fatal error
The <b>include()</b> function is mostly used when the file is not required and the application should continue to execute its process when the file is not found.	The <b>require()</b> function is mostly used when the file is mandatory for the application.
The <b>include()</b> function will only produce a warning ( <u>E_WARNING</u> ) and the script will continue to execute.	The <b>require()</b> will produce a fatal error (E_COMPILE_ERROR) along with the warning

# include vs. require

- Use the PHP `include_once` statement to load a file once.
- PHP `require_once` is the counterpart of the `include_once` except that the `require_once` issues an error if it fails to load the file.
- Also, the `require_once` won't load the file again if the file has been loaded.
- The `require` and `require_once`, `include`, `include_once` are language constructs, not functions.
- `include_once 'path_to_file';`
- `require_once 'path_to_file';`

# Uses of File in Applications

- Following are some of the practical use cases where we need files in our web applications.
  - Oftenly data is stored in form of JSON files, and Php code has to read the file and then display the data on web page.
  - In some simple applications, no database is used, rather data is stored in files.
  - In some web applications, you may have to prepare a file for user to download, in that case Php code will create a file, write data in it and then allow the user to download it.

# PHP — Files & I/O

- PHP File System allows us to create file, read file line by line, read file character by character, write file, append file, delete file and close file.
  - Opening a file
  - Reading a file
  - Writing a file
  - Closing a file

# Open File

- fopen() function is used to open file or URL and returns resource.

```
fopen ( string $filename , string $mode )
```

- If the fopen() function is successful, it returns a file handle, which can be used for further interaction with the file.
- This file handle is used by the fread(), fwrite(), fclose() function

```
<?php  
$handle = fopen("c:\\folder\\file.txt", "r");  
?>
```

# File modes

Mode	Description
r	Opens file in <b>read-only</b> mode. It places the file pointer at the beginning of the file.
r+	Opens file in <b>read-write</b> mode. It places the file pointer at the beginning of the file.
w	Opens file in <b>write-only</b> mode. It places the file pointer to the beginning of the file and truncates the file to zero length. If file is not found, it creates a new file.
w+	Opens file in <b>read-write</b> mode. It places the file pointer to the beginning of the file and truncates the file to zero length. If file is not found, it creates a new file.
a	Opens file in <b>write-only</b> mode. It places the file pointer to the end of the file. If file is not found, it creates a new file.
a+	Opens file in <b>read-write</b> mode. It places the file pointer to the end of the file. If file is not found, it creates a new file.
x	Creates and opens file in <b>write-only</b> mode. It places the file pointer at the beginning of the file. If file is found, <code>fopen()</code> function returns <code>FALSE</code> .
x+	It is same as x but it creates and opens file in <b>read-write</b> mode.
c	Opens file in <b>write-only</b> mode. If the file does not exist, it is created. If it exists, it is neither truncated (as opposed to 'w'), nor the call to this function fails (as is the case with 'x'). The file pointer is positioned on the beginning of the file
c+	It is same as c but it opens file in <b>read-write</b> mode.



# fclose()

- file is closed using fclose() function.

```
<?php
```

```
$file = fopen("demo.txt", 'r');
```

```
//some code to be executed
```

```
fclose($file);
```

```
?>
```

# File\_exists()

```
<?php
$filename = 'readme.txt';
if (file_exists($filename)) {
    $message = "The file $filename exists";
} else {
    $message = "The file $filename does not exist";
}
echo $message;
?>
```

# is\_file()

```
<?php
```

```
$filename = 'readme.txt';
```

```
if (is_file($filename)) {
```

```
    $message = "The file $filename exists";
```

```
}
```

```
else {
```

```
    $message = "The file $filename does not exist";
```

```
}
```

```
echo $message;
```


```
?>
```

# is\_readable()

```
<?php
$filename = 'readme.txt';
if (is_readable($filename)) {
    $message = "The file $filename exists and
readable ";
} else {
    $message = "The file $filename is not
readable";
}
echo $message;
?>
```

# is\_writable()

```
<?php
$filename = 'readme.txt';
if (is_writable($filename)) {
    $message = "The file $filename exists and
writable";
} else {
    $message = "The file $filename is not
writable";
}
echo $message;
?>
```



```
<?php
$filename = 'readme.txt';
if (!file_exists($filename)) {
    die("The file $filename does not exist.");
}
$f = fopen($filename, 'r');
if ($f) {
    // process the file
    // ...
    echo 'The file ' . $filename . ' is open';
    fclose($f);
}
?>
```

# PHP Read File

- PHP read functions to read data from file.
- Read all file data, read data line by line and read data character by character.
  - `fread()`
  - `fgets()`
  - `fgetc()`



# PHP Read File - fread()

```
string fread (resource $handle , int $length )
```

```
<?php
$filename = "c:\\file1.txt";
$fp = fopen($filename, "r");//open file in read mode
if( $fp == false )
{
    echo ( "Error in opening file" );
    exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );//read file
//Echo nl2br($filetext);
    echo "<pre>$filetext</pre>";//printing data of file
fclose($fp);//close file
?>
```

File size : 39 bytesThis is a test T  
paragraph

This is a test

This is second paragraph

# Read some characters from a file

```
<?php
$filename = './public/population.txt';
$f = fopen($filename, 'r');
if ($f) {
    $contents = fread($f, 100);
    fclose($f);
    echo nl2br($contents);
}
?>
```

# PHP Read File - fgets()

- The PHP fgets() function is used to read single line from the file.


```
string fgets ( resource $handle [,int $length ] )
```

```
<?php
$file = fopen("test.txt","r");

while(! feof($file))
{
    $line = fgets($file);
    echo $line. "<br>";
}

fclose($file);
?>
```

This is a simple test  
This is added text  
This is a test



```
<?php
$filename = './public/population.txt';
$lines = [ ];
$f = fopen($filename, 'r');
while (!feof($f)) {
    $lines[ ] = fgets($f);
}
print_r($lines);
fclose($f);
?>
```

# PHP Read File - fgetc()

- used to read single character from the file.
- To get all data using fgetc() function, use !feof() function inside the while loop.

```
string fgetc ( resource $handle )
```

```
<?php
```

```
$fp = fopen("c:\\file1.txt", "r");
```

```
while(!feof($fp)) {
```

```
    $ch= fgetc($fp);
```

```
    echo $ch;
```

```
}
```

```
fclose($fp);
```

```
?>
```

this is first line this is another line this is third line

# PHP Write File

- PHP fwrite() and fputs() functions are used to write data into file.
- To write data into file, you need to use w, r+, w+ mode.
- The PHP fwrite() function is used to write content of the string into file.

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

- New file can be created or text can be appended to an existing file
- Arguments for fwrite() function are file pointer and text that is to be written to file.

# PHP fwrite()

```
<?php
$fp = fopen('data.txt', 'w');
if( $fp == false )
{
    echo ( "Error in opening file" );
    exit();
}
fwrite($fp, 'welcome ');
fwrite($fp, 'to php file write');
fclose($fp);
echo "File written successfully";
?>
```



```
<?php
$fp = fopen('data.txt','a');//opens file in append mode
if( $fp == false )
{
echo ( "Error in opening file" );
exit();
}
fwrite($fp,' this is additional text ');
fwrite($fp,'appending data');
fclose($fp);

echo "File appended successfully";
?>
```

# Copy

```
<?php
$source = 'readme.txt';
$dest = 'readme.bak';
$flag=copy($source, $dest);
if($flag==true)
    echo "copied";
else
    echo "not copied";
?>
```

# PHP Delete File

- In PHP, we can delete any file using unlink() function.
- The unlink() function accepts one argument only: file name

```
bool unlink ( string $filename [, resource $context ] )
```

```
<?php
$status=unlink('data.txt');
if($status){
    echo "File deleted successfully";
}else{
    echo "Sorry!";
}
?>
```

# rename

```
<?php
$oldname = 'readme.txt';
$newname = 'readme_v2.txt';
if (rename($oldname, $newname)) {
    $message = 'The file was renamed successfully!';
} else {
    $message = 'There was an error renaming file; }
echo $message;
?>
```

## An Alternative Method Of Reading Data From A File: `file()` and `file_get_contents()` Function

- `file()` function: reads the entire file into an array with one line of code.
- Each element of the array then contains one line from the file.
- To display the contents of the file, simply iterate over the array in a `foreach()` loop and print each element.
- If don't want the data in an array: Try the `file_get_contents()` function, which reads the entire file into a string.

# Example: File()

```
<?php
// set file to read
$file = 'test.txt' or die('Could not open file!');
// read file into array
$data = file($file) or die('Could not read file!');
// loop through array and print each line
foreach ($data as $line) {
    echo $line;
}
?>
```

# Example:(file\_get\_contents())

```
<?php
// set file to read
$file = 'test.txt' ;
// read file into string
$data = file_get_contents($file) or die('Could not read file!');
// print contents
echo $data;
?>
```





•**Reading Remote Files:** Both **file\_get\_contents()** and **file()** support reading data from URLs, using either the HTTP or FTP protocol. It reads an HTML file of the Web into an array.

```
<?php
```

```
//read file into array
```

```
$arr= file ("https://www.google.com") or die ("ERROR: cannot find  
file");
```

```
foreach($arr as $line)
```

```
{
```

```
echo $line;
```

```
}
```

```
?>
```

In case of slow network links, it is more efficient to read a remote file in “chunks,” to maximize the efficiency of available network bandwidth. **fgets()** function can read a specific number of bytes from a file.

```
<?php
```

```
//read file into array (chunks)
```

```
$str="";
```

```
$file=fopen("http://www.google.com") or die("ERROR: cannot open  
file");
```

```
while(!feof($file))
```

```
{
```

```
$str=fgets($file,512);
```

```
//it opens only 512 bytes of data
```

```
}
```

```
fclose($file);
```

```
echo $str;
```

```
?>
```

# Writing To A File via file\_put\_contents()

- file\_put\_contents() function: which takes a string and writes it to a file in a single line of code.
- Example:

```
<?php
```

```
// set file to write
```


```
$filename = 'dump.txt';
```

```
// write to file
```


```
file_put_contents($filename, "Contents are added to the  
file ") or
```

```
die('Could not write to file');
```

```
?>
```



```
<?php
// get file path
if (file_exists("test6.txt"))
{
echo "file path: ". realpath("test6.txt");
} else {
echo "Named file does not exist. ";
}
?>
```



```
<?php
// get file information
if (file_exists("demo.txt"))
{
print_r(stat("demo.txt"));
}
Else
{
echo "Named file does not exist. ";
}
?>
```

# To check the status of the file

- PHP has lots of functions that allow you to test the status of a file like to check whether file exists, whether it's empty, whether it's readable or writable, and whether it's a binary or text file.
- `is_dir()` - returns a Boolean indicating whether the specified path is a directory
- `is_file()` - returns a Boolean indicating whether the specified file is a regular file
- `is_link()` - returns a Boolean indicating whether the specified file is a symbolic link
- `is_executable()` - returns a Boolean indicating whether the specified file is executable
- `is_readable()` - returns a Boolean indicating whether the specified file is readable
- `is_writable()` - returns a Boolean indicating whether the specified file is writable
- `filesize()` - gets size of file
- `filemtime()` - gets last modification time of file
- `fileatime()` - gets last access time of file
- `fileowner()` - gets file owner
- `filegroup()` - gets file group
- `fileperms()` - gets file permissions
- `filetype()` - gets file type

# Example:File operations

<html>

<head>

</head>

<body>

<form action="file\_oper.php" method="post">

Enter file path

<input type="text" name="file">

<input type="submit">

</form>

</body>

</html>

```
if (isset($_POST['file'])) {  
    echo 'File name: <b>'.$_POST['file'].'</b><br />';  
    /* check if file exists and display appropriate message */  
    if (file_exists($_POST['file'])) {  
        // print file size  
        echo 'File size: '.filesize($_POST['file']).' bytes<br />';  
        // print file owner  
        echo 'File owner: '.fileowner($_POST['file']).'<br />';  
        // print file group  
        echo 'File group: '.filegroup($_POST['file']).'<br />';  
        // print file permissions  
        echo 'File permissions: '.fileperms($_POST['file']).'<br />';  
        // print file type  
        echo 'File type: '.filetype($_POST['file']).'<br />';  
        // print file last access time  
        echo 'File last accessed on: '.date('Y-m-d', fileatime($_POST['file'])).'<br />';  
        // print file last modification time  
        echo 'File last modified on: '.date('Y-m-d', filemtime($_POST['file'])).'<br />';
```



```
if (is_dir($_POST['file'])) { // is it a directory?
echo 'File is a directory <br />'; }
if (is_file($_POST['file'])) { // is it a file?
echo 'File is a regular file <br />'; }
if (is_link($_POST['file'])) { // is it a link?
echo 'File is a symbolic link <br />'; }
if (is_executable($_POST['file'])) { // is it executable?
echo 'File is executable <br />'; }
if (is_readable($_POST['file'])) { // is it readable?
echo 'File is readable <br />'; }
if (is_writable($_POST['file'])) { // is it writable?
echo 'File is writable <br />'; }
}
else { echo 'File does not exist! <br />';
}
}
else { echo "file name not entered"; }
?>
```

# Global Variables - Superglobals

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- The PHP superglobal variables are:
  - **\$GLOBALS**
  - `$_SERVER`
  - **\$\_REQUEST**
  - **\$\_POST**
  - **\$\_GET**
  - `$_FILES`
  - `$_ENV`
  - **\$\_COOKIE**
  - **\$\_SESSION**

## PHP Superglobals

Variable	Description
<code>\$GLOBALS</code>	Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.
<code>\$_SERVER</code>	This is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these. See next section for a complete list of all the SERVER variables.
<code>\$_GET</code>	An associative array of variables passed to the current script via the HTTP GET method.
<code>\$_POST</code>	An associative array of variables passed to the current script via the HTTP POST method.
<code>\$_FILES</code>	An associative array of items uploaded to the current script via the HTTP POST method.
<code>\$_REQUEST</code>	An associative array consisting of the contents of <code>\$_GET</code> , <code>\$_POST</code> , and <code>\$_COOKIE</code> .
<code>\$_COOKIE</code>	An associative array of variables passed to the current script via HTTP cookies.
<code>\$_SESSION</code>	An associative array containing session variables available to the current script.
<code>\$_PHP_SELF</code>	A string containing PHP script file name in which it is called.
<code>\$php_errormsg</code>	<code>\$php_errormsg</code> is a variable containing the text of the last error message generated by PHP.

# PHP \$GLOBALS

- **\$GLOBALS** is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called **\$GLOBALS[index]**.
- The index holds the name of the variable.

```
<?php
    $x = 75;
    $y = 25;

    function addition() {
        $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
    }

    addition();
    echo $z;
?>
```

# PHP \$\_POST

- The PHP superglobals \$\_GET and \$\_POST are used to collect form-data.
- Form1.html

```
<!DOCTYPE HTML>
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

---

Name:

E-mail:

- welcome.php

```
<html>
<body>
<?php
echo "Welcome:" .$_POST["name"];
echo "<br>";
echo "Your email address is: " .$_POST["email"];
?>
</body>
</html>
```

Welcome abc

Your email address is: abc@gmail.com

# PHP \$\_GET

- Form.html

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<body>
```

```
<form action="welcome_get.php" method="get">
```

```
Name: <input type="text" name="name"><br>
```

```
E-mail: <input type="text" name="email"><br>
```

```
<input type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

Name: abc

E-mail: abc@gmail.com

Submit

- "welcome\_get.php

```
<html>
```

```
<body>
```

```
<?php
```

```
echo "Welcome:" .$_GET["name"]."<br>";
```

```
echo "Your email address is: " .$_GET["email"];
```

```
?>
```

```
</body>
```

```
</html>
```

Welcome abc

Your email address is: abc@gmail.com



# PHP \$\_REQUEST

welcome\_req.php

```
<html>
```

```
<body>
```

```
<?php
```

```
echo "Welcome:" .$_REQUEST["name"]."<br>";
```

```
echo "Your email address is: " .$_REQUEST["email"];
```

```
?>
```

```
</body>
```

```
</html>
```



# PHP \$ GET

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

```
<!DOCTYPE html >
```

```
<html>
```

```
<head>
```

```
<body>
```

```
<form action="script.php" method="get">
```

```
//<form action="<?php $_PHP_SELF ?>" method="GET">
```

```
//<form action="<?php echo $_SERVER['PHP_SELF'];?> method="get" >
```

```
User: <input type="text" name="user" /><br />
```

```
<input type="radio" name="gen" value="man" />Man
```

```
<input type="radio" name="gen" value="woman" />Woman<br />
```

```
<input type="submit" value="Send" />
```

```
</form>
```

```
</body>
```

```
</html>
```

- After the user fills the form and clicks the "submit" (Send) button, the URL sent to the server could look something like this:

http://PHP-programma/script.php?user=XYZ&gen=man

The "script.php" file can use the \$\_GET variable /array to collect form data.

```
<?php
if (isset($_GET['user']) && isset($_GET['gen']))
{
    $user = $_GET['user'];
    $gen = $_GET['gen'];
    echo 'User: '. $user. ' - gender: '. $gen;
}
?>
```

*\*\*\*Before using data received through get (or post), is indicated to check if they were set /sent (using the "isset()" function).*

POST	GET
<code>\$_POST</code> is an array of variables passed to the current script via the HTTP POST method.	<code>\$_GET</code> is an array of variables passed to the current script via the URL parameters.
Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request)	Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL).
POST has no limits on the amount of information to send.	GET also has limits on the amount of information to send. Limit is about 1024 characters
The variables are not displayed in the URL, it is not possible to bookmark the page	The variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases
POST from form submission	data sent by GET method can be accessed using <code>QUERY_STRING</code> environment variable.
Developers prefer POST for sending form data. The POST method can be used to send ASCII as well as binary data.	GET may be used for sending non-sensitive data. GET can't be used to send binary data, like images or word documents, to the server. *GET should NEVER be used for sending passwords or other sensitive information!

**`$_REQUEST` is a merging of `$_GET` and `$_POST` where `$_POST` overrides `$_GET`. Good to use `$_REQUEST` on self referential forms for validations.**

# PHP - A Simple HTML Form

```
<html>
```

```
<body>
```

```
<form action="nextpage.php" method="post">
```

```
Name: <input type="text" name="name"><br>
```

```
E-mail: <input type="text" name="email"><br>
```

```
<input type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

- "nextpage.php"

```
<html>
```

```
<body>
```

```
Welcome <?php echo $_POST["name"]; ?><br>
```

```
Your email address is <?php echo $_POST["email"]; ?>
```

```
</body>
```

```
</html>
```

**O/P:**

**Welcome Madhvi**

**Your email address is**

**madhvi.ramrakhiyani@gmail.com**

Try get method in this example

# The \$\_REQUEST variable

- The PHP \$\_REQUEST variable contains the contents of both \$\_GET, \$\_POST, and \$\_COOKIE.

```
<?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>
```

- <!DOCTYPE html>

<html>

<body>

< form method="post" action="<?php echo \$\_SERVER['PHP\_SELF'];?>" >

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

  <input type="submit">

</form>

<?php

if (\$\_SERVER["REQUEST\_METHOD"] == "POST") {

  // collect value of input field

  \$name = \$\_POST['fname'];

  if (empty(\$name)) {

    echo "Name is empty";

  } else {

    echo \$name;

  }

}

?>

</body>

</html>

# **\$\_SERVER**

Element/Code	Description
<code>\$_SERVER['PHP_SELF']</code>	Returns the filename of the currently executing script
<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as HTTP/1.1)
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (such as POST)
<code>\$_SERVER['REQUEST_TIME']</code>	Returns the timestamp of the start of the request (such as 1377687496)
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script





```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
echo "<br>"
?>
```



# File upload

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the `file_uploads` directive, and set it to On:

```
file_uploads = On
```

- Some rules to follow for the HTML form above:
- Make sure that the form uses `method="post"`
- The form also needs the following attribute: `enctype="multipart/form-data"`. It specifies which content-type to use when submitting the form

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<form action="upload.php" method="post" enctype="multipart/form-data">
```

Select image to upload:

```
<input type="file" name="fileToUpload" id="fileToUpload">
```

```
<input type="submit" value="Upload Image" name="submit">
```

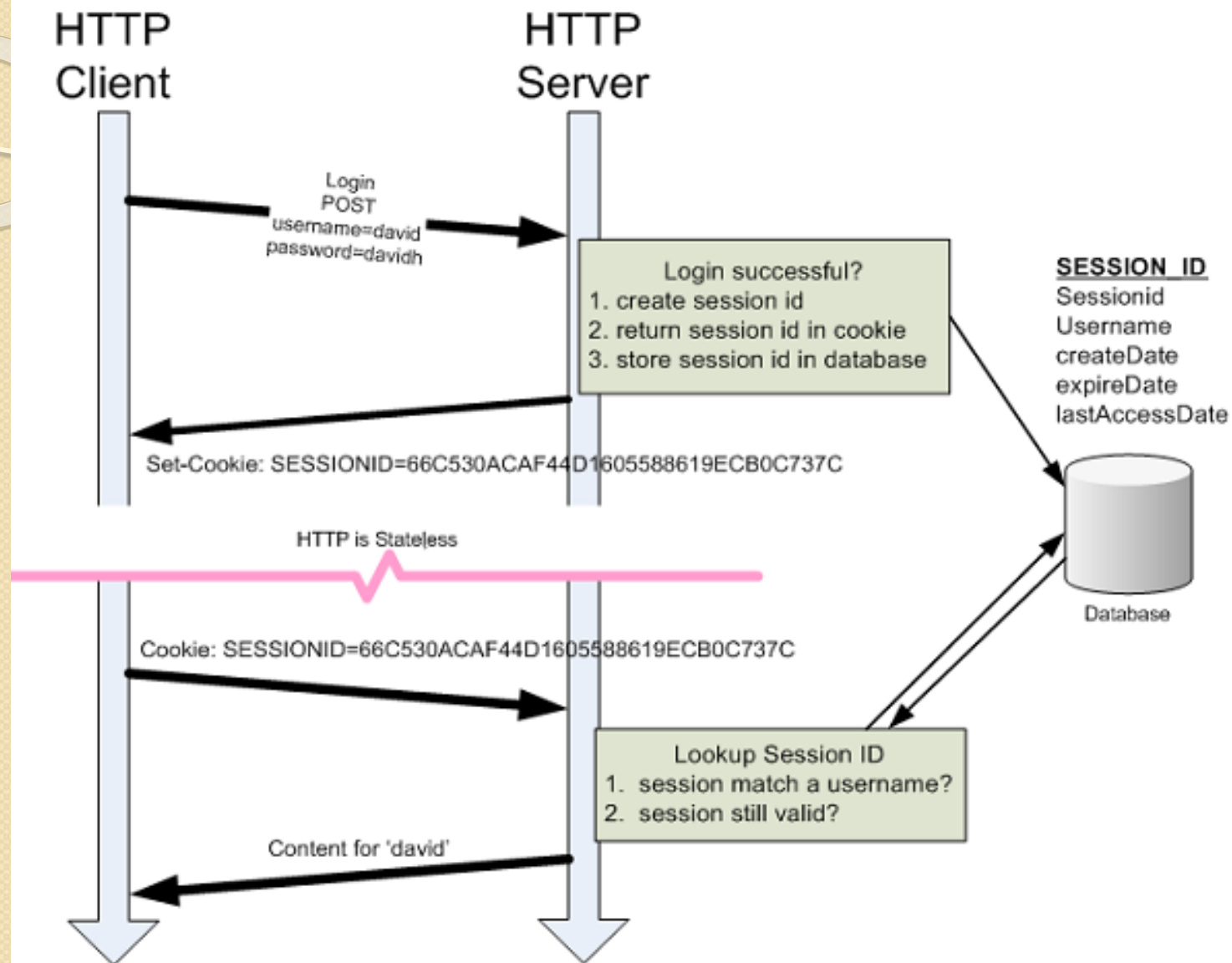
```
</form>
```

```
</body>
```

```
</html>
```

- 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 following five variables –
- **\$\_FILES['file']['tmp\_name']** – the uploaded file in the temporary directory on the 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.

# Session and cookie

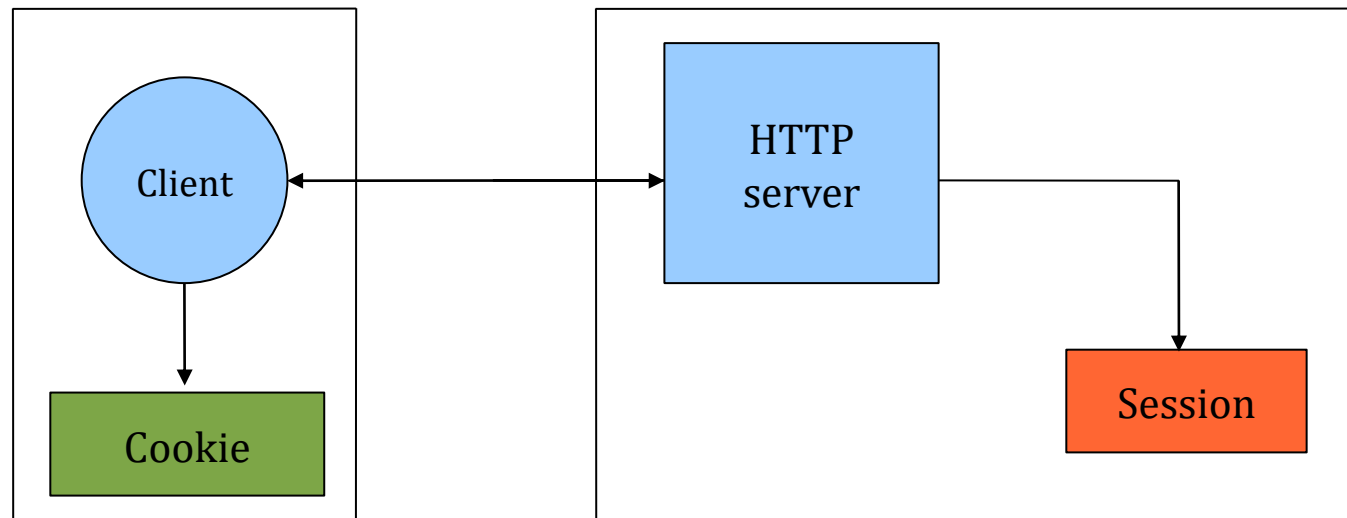


# Persistence and HTTP

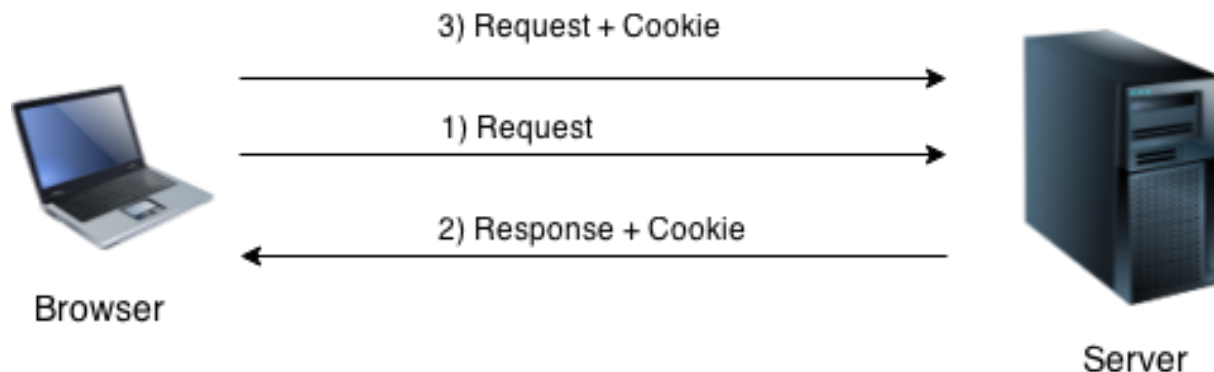
Recall http is a stateless protocol. It remembers nothing about previous transfers


Two ways to achieve persistence:

- PHP cookies
- PHP sessions



- Cookie is created at server side and saved to client browser. Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.
- cookie can be created, sent and received at server end.



- 
- **Why and when to use Cookies?**
  - Http is a stateless protocol; cookies allow us to track the state of the application using small files stored on the user's computer. The path where the cookies are stored depends on the browser. Internet Explorer usually stores them in Temporal Internet Files folder.
  - Personalizing the user experience – this is achieved by allowing users to select their preferences. The page requested that follows are personalized based on the set preferences in the cookies.
  - Tracking the pages visited by a user

# Session and Cookie

## Cookie

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

### Create Cookies With PHP

- A cookie is created with the `setcookie()` function.
- **Syntax**  
`setcookie(name, value, expire, path, domain, secure, httponly);`
- Only the *name* and *value* parameters are required. All other parameters are optional.
- There can be no HTML, text or white-space output before calling the **`setcookie()` function**.



# setcookie(name,value,expire,path,domain,secure)

Parameter	Description
<b>name</b>	(Required). Specifies the name of the cookie is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
<b>value</b>	(Required). Specifies the value of the cookie. is the content that you actually want to store.
<b>expire</b>	(Optional). Specifies when the cookie expires. e.g. <b>time()+3600*24*30</b> will set the cookie to expire in <b>30 days</b> . If this parameter is not set, the cookie will expire at the end of the session (when the browser closes).
<b>path</b>	(Optional). directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories If set to <b>/</b> , the cookie will be available within the entire domain. If set to <b>/phptest/</b> , the cookie will only be available within the test directory and all sub-directories of <b>phptest</b> . The default value is the current directory that the cookie is being set in.
<b>domain</b>	(Optional). Specifies the domain name of the cookie. To make the cookie available on all subdomains of example.com then you'd set it to ".example.com". Setting it to www.example.com will make the cookie only available in the www subdomain
<b>secure</b>	(Optional). Specifies whether or not the cookie should only be transmitted over a secure <b>HTTPS</b> connection. <b>TRUE</b> indicates that the <u>cookie will only be set</u> if a secure connection exists. Default is <b>FALSE</b> .



```
<?php
    $cookie_name = "name";
    $cookie_value = "XYZ";
    setcookie($cookie_name, $cookie_value, time() +
(86400 * 30));      // 86400 = 1 day
?>
```

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

```
<?php

    if(!isset($_COOKIE[$cookie_name]))
    {
        echo "Cookie named '" . $cookie_name . "' is not set!";
    }
    else
    {
        echo "Cookie '" . $cookie_name . "' is set!<br>";
        echo "Value is: " . $_COOKIE[$cookie_name];
    }
?>
</body>
</html>
```

## Modify a Cookie Value

- To modify a cookie, just set (again) the cookie using the `setcookie()` function

## Delete a Cookie

- To delete a cookie, use the `setcookie()` function with an expiration date in the past:

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

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

```
<?php
echo "Cookie 'user' is deleted.";
?>
```

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

# Deleting a cookie

- Set the cookie with its name only:

```
setcookie ("mycookie") ;
```

## Check if Cookies are Enabled

- First, try to create a test cookie with the setcookie() function, then use count on \$\_COOKIE array variable:

```
<?php
    setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
```

```

    <?php
    if(count($_COOKIE) > 0)
    {
        echo "Cookies are enabled.";
    }
else {
    echo "Cookies are disabled.";
}
?>
```

```

</body>
</html>
```

# PHP sessions

- By default, HTML and web servers don't keep track of information entered on a page when the client's browser opens another page. Thus, doing anything involving the same information across several pages can sometimes be difficult.
- *Sessions* help solve this problem by maintaining data during a user's visit, and can store data that can be accessed from page to page in your site.
- You can use session variables for storing information (this is one way that a "shopping cart" function can work for an online shop, for example).
- Servers keep track of users' sessions by using a session identifier, which is generated by the server when a session starts and is then used by the browser when it requests a page from the server. This session ID can be sent through a cookie (the default behavior) or by passing the session ID in the URL string.
- Sessions only store information temporarily, so if you need to preserve information, say, between visits to the same site, you should likely consider a method such as using a cookie or a database to store such information.

# PHP 5 Sessions

- PHP session is used to store and pass information from one page to another temporarily (until user close the website).
- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- The web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables hold information about one single user, and are available to all pages in one application.


# PHP Sessions

You can store user information (e.g. username, items selected, etc.) in the **server side** for later use using PHP session.

**Sessions** work by creating a unique id (**UID**) for each visitor and storing variables based on this **UID**.

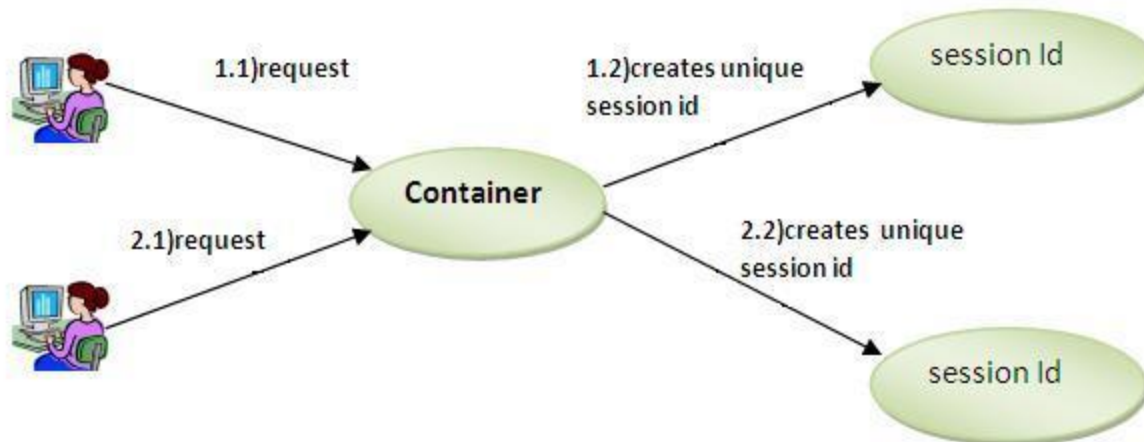
The **UID** is either stored in a **cookie** or is **propagated in the URL**.



- 
- 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`



- PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.
- PHP session creates unique user id for each browser to recognize the user and avoid conflict between multiple browsers.



# When should you use sessions?

- Need for data to be stored on the server
- Unique session information for each user
- Transient data, only relevant for short time
- Data does not contain secret information
- Similar to Cookies, but it is stored on the server
- More secure, once established, no data is sent back and forth between the machines
- Works even if cookies are disabled
- Example: we want to count the number of “hits” on our web page.

Before you can store user information in your PHP session, you must first start up the session.

**session\_start()** function must appear BEFORE the **<html>** tag.

```
<?php session_start(); ?>
```

```
<html>
```

```
<body>
```

```
</body>
```

```
</html>
```

# PHP Sessions

- **When a session is started following things happen**
  1. PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as **3c7foj34c3jj973hjkop2fc937e3443**.
  2. A cookie called **PHPSESSID** is automatically sent to the **user's computer to store unique** session identification string.
  3. 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**.
  4. 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.
  5. 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.

## Start a PHP Session

- PHP `session_start()` function is used to start the session. It starts a new or resumes existing session. It returns existing session if session is created already. If session is not available, it creates and returns new session.
- Session variables are stored in associative array called **`$_SESSION[]`**. **These variables can** be accessed during lifetime of a session.

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

**Note:** The `session_start()` function must be the very first thing in your document. Before any HTML tags.


# Get PHP Session Variable Values

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
```

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

Favorite color is green.  
Favorite animal is cat.



```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
print_r($_SESSION);
?>
```

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

```
Array ( [favcolor] => green [favanimal] => cat )
```



## Modify a PHP Session Variable

- To change a session variable, just overwrite it:

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
```

```

</body>
</html>
```



# Destroy a PHP Session

```
<?php
    session_start();
    ?>
<!DOCTYPE html>
<html>
<body>

    <?php
    // remove all session variables
    session_unset();

    // destroy the session
    session_destroy();
    echo "All session variables are now removed, and the session is
    destroyed."
    ?>

</body>
</html>
```

## More on session variables

- You need to include a call to the `session_start()` function for each page on which you want to access the session variables.
- A session will end once you quit the browser (unless you've set appropriate cookies that will persist), or you can call the `session_destroy()` function. (Note, however, even after calling the `session_destroy()` function, session variables are still available to the rest of the currently executing PHP page.)
- The function `session_unset()` removes all session variables. If you want to remove one variable, use the `unset($var)` function call.
- The default timeout for session files is 24 minutes. It's possible to change this timeout.

# PHP Session Counter Example

```
<?php
    session_start();

    if (!isset($_SESSION['counter'])) {
        $_SESSION['counter'] = 1;
    } else {
        $_SESSION['counter']++;
    }
    echo ("Page Views: ".$_SESSION['counter']);
?>
```

Session	Cookie
Sessions are server-side files that contain user information	Cookies are client-side files that contain user information as text file format
Session Max life time is 1440 Seconds(24 Minutes) as defined in php.ini file	We have to set cookie max life time manually with php code with setcookie function.
It stores unlimited amount of data.	It stores limit amount of data. It is only allowing 4kb[4096 bytes]
It is holding the multiple variable in sessions.	It is not holding the multiple variable in cookies.
http://php.net/session.gc-maxlifetime session.gc_maxlifetime = 1440	setcookie("email", 'test@example.com', time()+3600); /* expire in 1 hour */
You can edit this value if you need custom session life.	In above example Cookie Name : email Cookie Value : <u>test@example.com</u> Expire time : 1 hour after current time (1 Hour = 3600 Seconds)
In php \$_SESSION super global variable is used to manage session.	In php \$_COOKIE super global variable is used to manage cookie.

# Form validation

## Empty string:

```
if (empty ($_POST["name"])) {  
    $errMsg = "Error! You didn't enter the Name."  
      
    echo $errMsg;  
} else {  
    $name = $_POST["name"];  
}
```

- **Validate String**

```
$name = $_POST ["Name"];  
if (!preg_match ("/^[a-zA-z]*$/", $name) )  
{  
    $ErrMsg = "Only alphabets and whitespace are  
allowed.";  
    echo $ErrMsg;  
}  
else {  
    echo $name;  
}
```

### **Validate Number:**

```
$mobilenumber = $_POST ["Mobile_no"];  
if (!preg_match ("/^[0-9]*$/", $mobilenumber) ){  
    $ErrMsg = "Only numeric value is allowed."  
    echo $ErrMsg;  
} else {  
    echo $mobilenumber;  
}
```

### **Length validation:**

```
$mobilenumber = $_POST ["Mobile"];  
$length = strlen($mobilenumber);  
  
if ( $length < 10 && $length > 10) {  
    $ErrMsg = "Mobile must have 10 digits."  
    echo $ErrMsg;  
} else {  
    echo "Your Mobile number is: " . $mobilenumber;  
}
```

# Email validation

```
$email = $_POST ["Email"];  
$pattern = "^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[a-z0-9-]+(\\.[a-z0-9-]+)*(\\.[a-z]{2,3})$^";  
if (!preg_match ($pattern, $email) ){  
    $ErrMsg = "Email is not valid.";  
    echo $ErrMsg;  
}  
else {  
    echo "Your valid email address is: " . $email;  
}  
  
OR  
  
$email = $_POST ["Email"];  
  
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {  
    echo("$email is a valid email address");  
}  
else {  
    echo("$email is not a valid email address");  
}
```



# Button Click Validate

validates that the user click on submit button and send the form data to the server one of the following method - get or post.

```
if (isset ($_POST['submit'])) {  
    echo "Submit button is clicked."  
    if ($_SERVER["REQUEST_METHOD"] == "POST") {  
        echo "Data is sent using POST method "  
    }  
}  
else {  
    echo "Data is not submitted";  
}
```