PROGRAMMING ON PHP

PROGRAMMING ON PHP

WEB CONCEPTS IN PHP

GET AND POST

We have two HTTP request methods in PHP for handling the forms, where submitted form-data from users can be collected using these methods. In order to send information to the web server from the browser client, we use GET and POST methods.

GET Method: Data is requested from a specific resource

POST Method: Data is submitted to be processed to a specific resource

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.

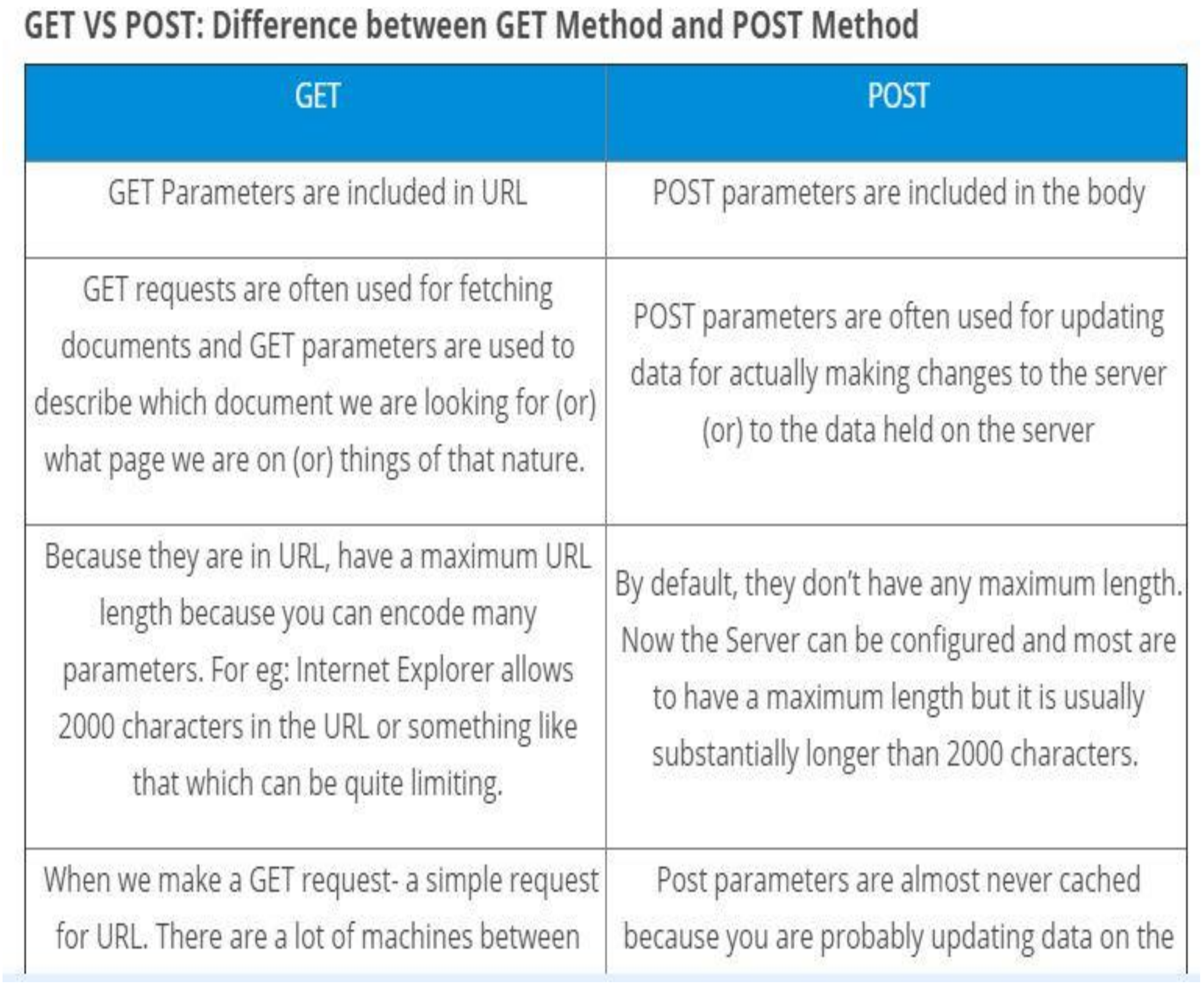
http://www.test.com/index.htm?name1=value1&name2=value2

```
<?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>
```

The POST Method

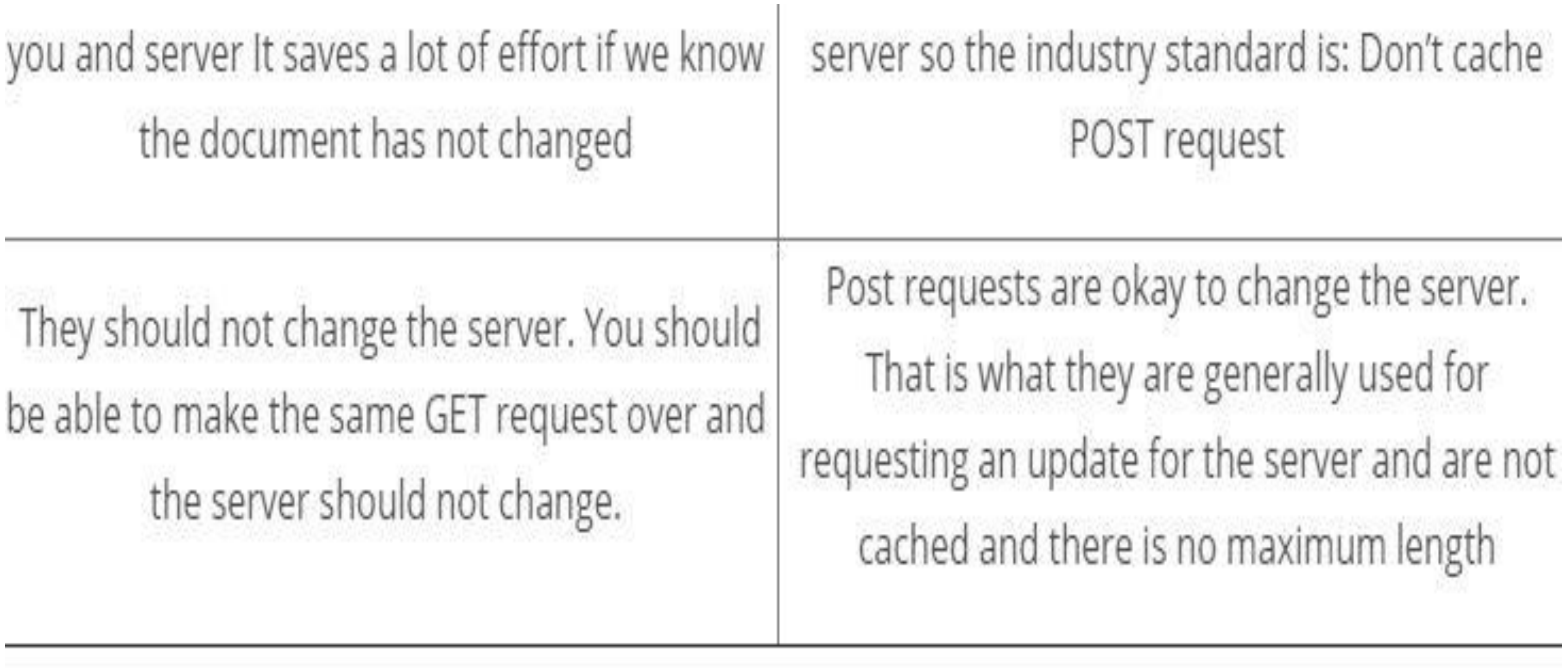
The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

```
<?php
 if( $_POST["name"] || $_POST["age"] )
{
 if (preg_match("/[^A-Za-z'-]/",$_POST['name'] ))
{
die ("invalid name and name should be alpha");
}
 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>
```



GET VS POST: Difference between GET Method and POST Method

GET	POST
GET Parameters are included in URL	POST parameters are included in the body
GET requests are often used for fetching documents and GET parameters are used to describe which document we are looking for (or) what page we are on (or) things of that nature.	POST parameters are often used for updating data for actually making changes to the server (or) to the data held on the server
Because they are in URL, have a maximum URL length because you can encode many parameters. For eg: Internet Explorer allows 2000 characters in the URL or something like that which can be quite limiting.	By default, they don't have any maximum length. Now the Server can be configured and most are to have a maximum length but it is usually substantially longer than 2000 characters.
When we make a GET request- a simple request for URL. There are a lot of machines between	Post parameters are almost never cached because you are probably updating data on the



you and server It saves a lot of effort if we know the document has not changed	server so the industry standard is: Don't cache POST request
They should not change the server. You should be able to make the same GET request over and the server should not change.	Post requests are okay to change the server. That is what they are generally used for requesting an update for the server and are not cached and there is no maximum length

The $_REQUEST variable

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

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

Try out 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>
```

FILE INCLUSION AND FILES & I/O

PHP - File Inclusion

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

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.

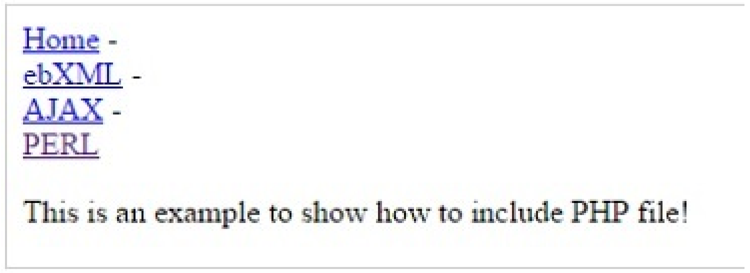
Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

<a href="http://www.tutorialspoint.com/index.htm">Home</a> - <a href="http://www.tutorialspoint.com/ebxml">ebXML</a> - <a href="http://www.tutorialspoint.com/ajax">AJAX</a> - <a href="http://www.tutorialspoint.com/perl">PERL</a> <br />

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

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

It will produce the following result −



Home -
ebXML -
AJAX -
PERL

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

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.

You can try using above example with require() function and it will generate same result. But if you will try 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>
```

Now lets 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>
```

PHP - FILES & I/O

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.

Sr.No	Mode & Purpose
1	**r** Opens the file for reading only. Places the file pointer at the beginning of the file.
2	**r+** Opens the file for reading and writing. Places the file pointer at the beginning of the file.
3	**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 files does not exist then it attempts to create a file.
4	**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 files does not exist then it attempts to create a file.
5	**a** Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
6	**a+** Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

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.

So here are the steps required to read a file with PHP.

Open a file using **fopen()** function.

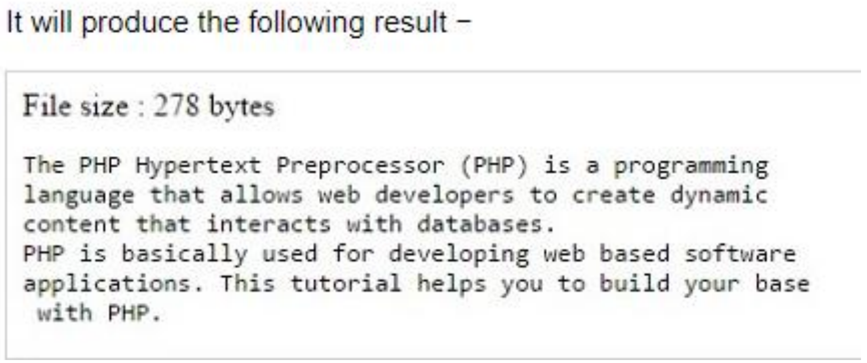
Get the file's length using **filesize()** function.

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 then displays those contents on the web page.

```
html>
 <head>
<title>Reading a file using PHP</title> </head>
<body>
<?php
$filename = "tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
echo ( "Error in opening file" );
exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );
fclose( $file );
echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>
</body>
 </html>
```



It will produce the following result −

```
File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming
language that allows web developers to create dynamic
content that interacts with databases.
PHP is basically used for developing web based software
applications. This tutorial helps you to build your base
 with PHP.
```

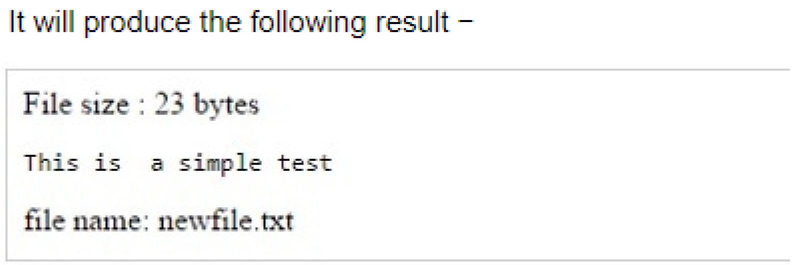
Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_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
$filename = "newfile.txt";
$file = fopen( $filename, "r" );
 if( $file == false )
 {
 echo ( "Error in opening file" );
exit();
}
```

```
$filesize = filesize( $filename );
 $filetext = fread( $file, $filesize );
fclose( $file );
 echo ( "File size : $filesize bytes" );
echo ( "$filetext" );
echo("file name: $filename");
 ?>
 </body>
 </html>
```



It will produce the following result −

```
File size : 23 bytes

This is  a simple test

file name: newfile.txt
```

FUNCTIONS

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.

PHP provides us with two major types of functions:

Built-in functions : PHP provides us with huge collection of built-in library functions. These functions are already coded and stored in form of functions. To use those we just need to call them as per our requirement like, var_dump, fopen(), print_r(), gettype() and so on.

User Defined Functions : Apart from the built-in functions, PHP allows us to create our own customised functions called the user-defined functions.
Using this we can create our own packages of code and use it wherever necessary by simply calling it.

Creating a Function

While creating a user defined function we need to keep few things in mind:

Any name ending with an open and closed parenthesis is a function.

A function name always begins with the keyword *function*.

To call a function we just need to write its name followed by the parenthesis

A function name cannot start with a number. It can start with an alphabet or underscore.

A function name is not case-sensitive.

Syntax:

```
function function_name()

{

 executable code;

}
```

```php
<?php

function funcGeek()
{
    echo "This is Geeks for Geeks";
}

// Calling the function
funcGeek();

?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

```php
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

USER DEFINED FUNCTIONS

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads. A function will be executed by a call to the function.

Function names are NOT case-sensitive.

Syntax

```
function functionName() {
    code to be executed;
}
```

EXAMPLE OF USER DEFINED FUNCTIONS

```
<html>
<body>
<?php
function writeMsg() {
    echo "Hello Nabraj!";
}
writeMsg();
?>
</body>
</html>
```

Output:

```
Hello Nabraj!
```

FUNCTION ARGUMENTS

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

FUNCTION ARGUMENTS

```
<html>
<body>
<?php
function familyName($fname) {
    echo "$fname<br>";
}
familyName("Nabraj");
familyName("Ramesh");
familyName("Pawan");
familyName("Nabin");
familyName("Khim");
?>
</body>
</html>
```

```
Output:
Nabraj
Ramesh
Pawan
Nabin
Khim
```

FUNCTIONS - RETURNING VALUES

To let a function return a value, use the return statement:

```
<html>
<body>
<?php
function sum(int $x, int $y) {
   $z = $x + $y;
   return $z;
}
echo sum(5,10);
echo "<br>";
echo sum(7,13);
?>
</body>
</html>
```

Output:

```
15
20
```

FUNCTION TO FIND AREA OF RECTANGLE AND RETURN THE VALUE

```
<html>
<body>
<?php
function sum( int $l, int $b) {
    $a = $l * $b;
            return $a;
}
$b= sum(15,200);
echo $b;
?>
</body>
</html>
```

FUNCTION TO ADD TWO NUMBER

```
<html>
<body>
<?php
function sum(int $x, int $y) {
   $z = $x + $y;
          echo $z;
}
echo sum(5,10);
?>
</body>
</html>
```

COOKIE

What is a cookie

Cookies are used to store the information of a web page in a remote browser, so that when the same user comes back to that page, that information can be retrieved from the browser itself.

USES OF COOKIE

Cookies are often used to perform following tasks:

Session management: Cookies are widely used to manage user sessions. For example, when you use an online shopping cart, you keep adding items in the cart and finally when you checkout, all of those items are added to the list of items you have purchased. This can be achieved using cookies.

User identification: Once a user visits a webpage, using cookies, that user can be remembered. And later on, depending upon the search/visit pattern of the user, content which the user likely to be visited are served. A good example of this is 'Retargetting'. A concept used in online marketing, where depending upon the user's choice of content, advertisements of the relevant product, which the user may buy, are served.

Tracking / Analytics: Cookies are used to track the user. Which, in turn, is used to analyze and serve various kind of data of great value, like location, technologies (e.g. browser, OS) form where the user visited, how long (s)he stayed on various pages etc.

HOW TO CREATE A COOKIE IN PHP

PHP has a setcookie() function to send a cookie. We will discuss this function in detail now.

Syntax:

```
        setcookie(name, value, expire)
```

Example to Create/Retrieve a Cookie:

```
<html>
<?php
$cookie_name = "user";
$cookie_value = "Nabraj Koirala";
setcookie($cookie_name, $cookie_value, time() + 10);
?>
<html>
<body>
<?php
if(!isset($_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>
```

PHP SESSIONS

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.

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

Start a PHP Session:

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

```
 <?php
// Start the session
session_start();
?>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

Get PHP Session Variable Values:

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

DESTROY A PHP SESSION

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

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

SENDING EMAILS

What is PHP mail?

PHP mail is the built in PHP function that is used to send emails from PHP scripts.

The mail function accepts the following parameters;

Email address

Subject

Message

CC or BC email addresses

Sending mail using PHP

The PHP mail function has the following basic syntax

```
<?php
mail($to_email_address,$subject,$message,[$headers],[$parameters]);

?>
```

HERE,

“$to_email_address” is the email address of the mail recipient

“$subject” is the email subject

“$message” is the message to be sent.

“[$headers]” is optional, it can be used to include information such as CC, BCC

- CC is the acronym for carbon copy. It’s used when you want to send a copy to an interested person i.e. a complaint email sent to a company can also be sent as CC to the complaints board.
- BCC is the acronym for blind carbon copy. It is similar to CC. The email addresses included in the BCC section will not be shown to the other recipients.

Simple Mail Transmission Protocol (SMTP)

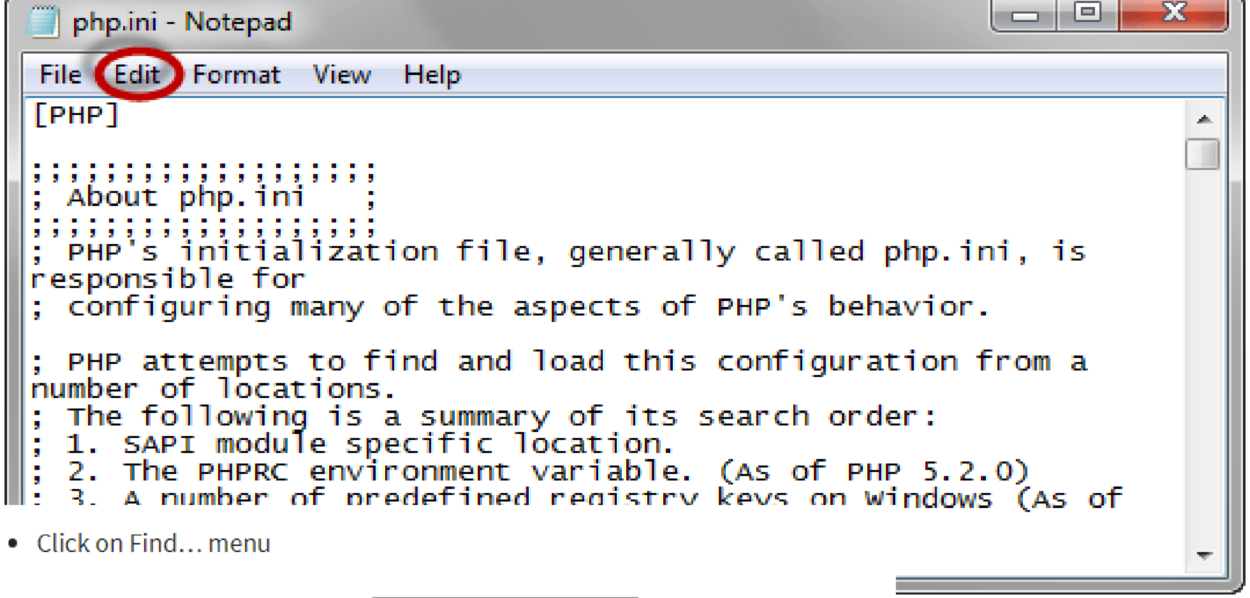
PHP mailer uses Simple Mail Transmission Protocol (SMTP) to send mail.

On a hosted server, the SMTP settings would have already been set.

The SMTP mail settings can be configured from “php.ini” file in the PHP installation folder.

Configuring SMTP settings on your localhost Assuming you are using xampp on windows, locate the “php.ini” in the directory “C:\xampp\php”.

Open it using notepad or any text editor. We will use notepad in this example. Click on the edit menu

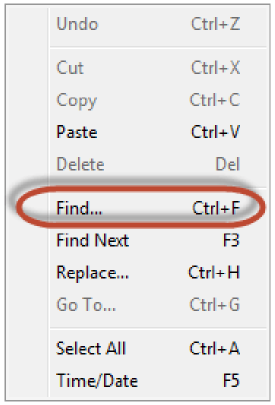


php.ini - Notepad

File Edit Format View Help

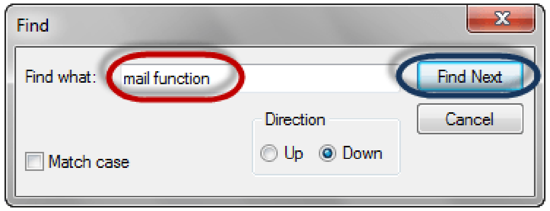
```
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; PHP's initialization file, generally called php.ini, is
responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a
number of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on Windows (As of
```

- Click on Find… menu



Undo	Ctrl+Z
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Find...	Ctrl+F
Find Next	F3
Replace...	Ctrl+H
Go To...	Ctrl+G
Select All	Ctrl+A
Time/Date	F5

- The find dialog menu will appear



Find

Find what: mail function

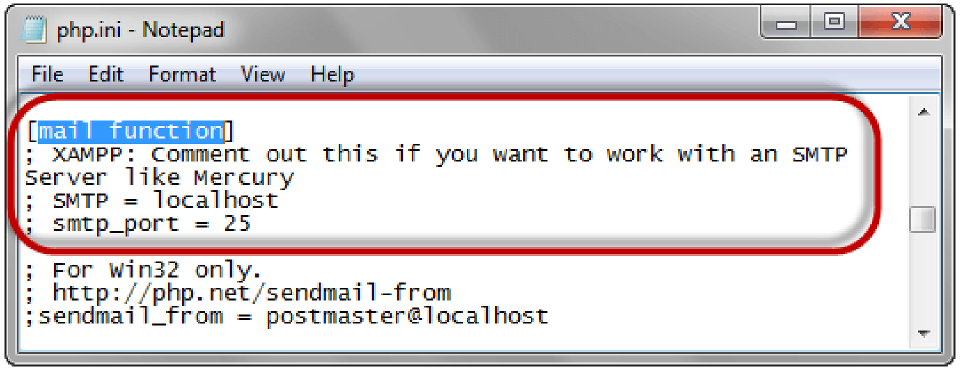
Find Next

Cancel

Direction: Up [✓ Down]

Match case

- Click on Find Next button



```
[mail function]
; XAMPP: Comment out this if you want to work with an SMTP
Server like Mercury
; SMTP = localhost
; smtp_port = 25

; For Win32 only.
; http://php.net/sendmail-from
;sendmail_from = postmaster@localhost
```

ocate the entries*[mail function]*

; XAMPP: Don’t remove the semi column if you want to work with an SMTP Server like Mercury

; SMTP = localhost

; smtp_port = 25

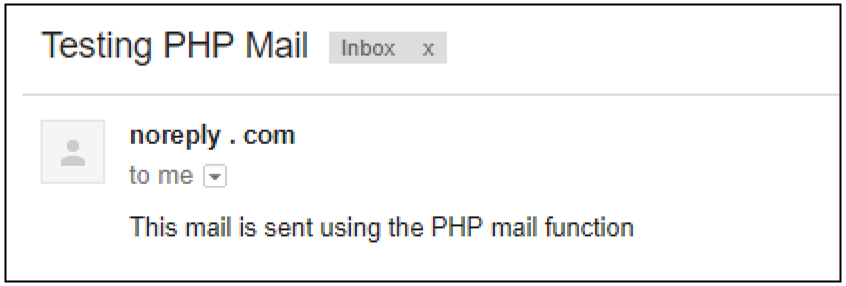
Remove the semi colons before SMTP and smtp_port and set the SMTP to your smtp server and the port to your smtp port. Your settings should look as follows

- SMTP = smtp.example.com
- smtp_port = 25
- *Note the SMTP settings can be gotten from your web hosting providers.*
- If the server requires authentication, then add the following lines.
    - auth_username = example_username@example.com
    - auth_password = example_password
    - Save the new changes.
    - Restart Apache server.

Let’s now look at an example that sends a simple mail.

```
<?php
 $to_email = 'name @ company . com';
$subject = 'Testing PHP Mail';
 $message = 'This mail is sent using the PHP mail function';
$headers = 'From: noreply @ company . com';
mail($to_email,$subject,$message,$headers);
?>
```

Output:



```
Testing PHP Mail  Inbox x
noreply . com
to me
This mail is sent using the PHP mail function
```

Note: the above example only takes the 4 mandatory parameters.

You should replace the above fictitious email address with a real email address.

FILE UPLOADING

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

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
```

CREATE THE HTML FORM

Next, create an HTML form that allow users to choose the image file they want to upload:

UPLOAD.PHP

```
<html>
<body>
<form method='post' enctype='multipart/form-data'
    action='receive.php'>
<input type='file' name='myfile' />
<input type='submit' name='upload' value='SEND' />
</form>
</body>
</html>
```

RECEIVE.PHP

```php
<?php
$name=$_FILES['myfile']['name'];
$tmp_name=$_FILES['myfile']['tmp_name'];
if(move_uploaded_file($tmp_name, $name))
{
    echo "file uploaded";
}
else
{
    echo "not successful";
}
?>
```

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

Without the requirements above, the file upload will not work.

Other things to notice:

The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

ERROR HANDLING

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.

some of the most common error checking methods in PHP.

Simple "die()" statements

Custom errors and error triggers

Error reporting

Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file:

Example

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

If the file does not exist you might get an error like this:

Warning: fopen(mytestfile.txt) [function.fopen]: failed to open stream:
No such file or directory in **C:\webfolder\test.php** on line **2**

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it:

Example

```
<?php
  if(file_exists("mytestfile.txt")) {
    $file = fopen("mytestfile.txt", "r");
  } else {
    die("Error: The file does not exist.");
  }
  ?>
```

Now if the file does not exist you get an error like this:

```
Error: The file does not exist.
```

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error.

However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

```
error_function(error_level,error_message,
    error_file,error_line,error_context)
```

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
1	E_ERROR	A fatal run-time error. Execution of the script is stopped
2	E_WARNING	A non-fatal run-time error. Execution of the script is not stopped
8	E_NOTICE	A run-time notice. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	A fatal user-generated error. This is like an E_ERROR, except it is generated by the PHP script using the function trigger_error()
512	E_USER_WARNING	A non-fatal user-generated warning. This is like an E_WARNING, except it is generated by the PHP script using the function trigger_error()
1024	E_USER_NOTICE	A user-generated notice. This is like an E_NOTICE, except it is generated by the PHP script using the function trigger_error()
2048	E_STRICT	Not strictly an error.
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Now lets create a function to handle errors:

```
function customError($errno, $errstr) {
    echo "<b>Error:</b> [$errno] $errstr<br>";
    echo "Ending Script";
    die();
  }
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

Set Error Handler

The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

set_error_handler("customError");

Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, a second parameter could be added to specify an error level.

Example

Testing the error handler by trying to output variable that does not exist:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr";
}

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

The output of the code above should be something like this:

Error: [8] Undefined variable: test

Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger_error() function.

Example

In this example an error occurs if the "test" variable is bigger than "1":

```
<?php
    $test=2;
    if ($test>=1) {
      trigger_error("Value must be 1 or below");
    }
    ?>
```

The output of the code above should be something like this:

```
Notice: Value must be 1 or below
    in C:\webfolder\test.php on line 6
```

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.

Possible error types:

E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted

E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted

E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

Example

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurs we will use our custom error handler and end the script:

```
<?php
    //error handler function
    function customError($errno, $errstr) {
      echo "<b>Error:</b> [$errno] $errstr<br>";
      echo "Ending Script";
      die();
    }

    //set error handler
    set_error_handler("customError",E_USER_WARNING);

    //trigger error
    $test=2;
    if ($test>=1) {
      trigger_error("Value must be 1 or below",E_USER_WARNING);
    }
    ?>
```

The output of the code above should be something like this:

```
Error: [512] Value must be 1 or below
    Ending Script
```

BUGS DEBUGGING

Debugging is the process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects.