

Chapter One - Server Side Scripting Basics

Introduction

Today's Web users expect exciting pages that are updated frequently and provide a customized experience. For them, Web sites are more like communities, to which they'll return time and again. At the same time, Web-site administrators want sites that are easier to update and maintain, understanding that's the only reasonable way to keep up with visitors' expectations. For these reasons and more, PHP and MySQL have become the de facto standards for creating dynamic, database driven Web sites.

What Are Dynamic Web Sites?

Dynamic Web sites are flexible and potent creatures, more accurately described as applications than merely sites. Dynamic Web sites

- Respond to different parameters (for example, the time of day or the version of the visitor's Web browser)
- Have a "memory," allowing for user registration and login, e-commerce, and similar processes
- Almost always integrate HTML forms, allowing visitors to perform searches, provide feedback, and so forth
- Often have interfaces where administrators can manage the site's content
- Are easier to maintain, upgrade, and build upon than statically made sites

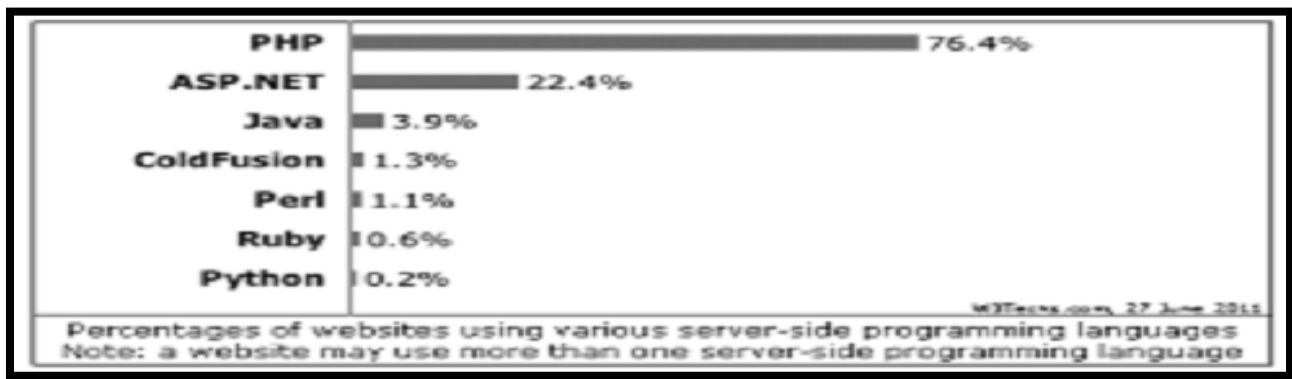
There are many technologies available for creating dynamic Web sites. The most common are ASP.NET (Active Server Pages, a Microsoft construct), JSP (Java Server Pages), ColdFusion, Ruby on Rails (a Web development framework for the Ruby programming language), and PHP. Dynamic Web sites don't always rely on a database, but more and more of them do, particularly as excellent database applications like MySQL are available at little to no cost.

What is PHP?

PHP originally stood for "Personal Home Page" as it was created in 1994 by Rasmus Lerdorf to track the visitors to his online résumé. As its usefulness and capabilities grew (and as it started being used in more professional situations), it came to mean "PHP: Hypertext Preprocessor." According to the official PHP Web site, found at www.php.net, PHP is a "widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML."

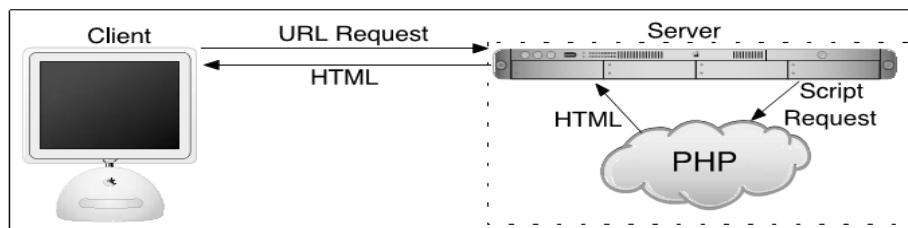
PHP is a server-side, cross-platform technology, both descriptions being important. Server side refers to the fact that everything PHP does occurs on the server. A Web server application, like Apache or Microsoft's IIS (Internet Information Services), is required and all PHP scripts must be accessed through a URL (<http://something>). Its cross-platform nature means that PHP runs on most operating systems, including Windows, UNIX (and its many variants), and Macintosh. More important, the PHP scripts written on one server will normally work on another with little or no modification.

PHP has seen an exponential growth in use since its inception, and is the server-side technology of choice on over 76 percent of all Web sites. In terms of all programming languages, PHP is the fifth most popular after Java, C, C++ and C#.

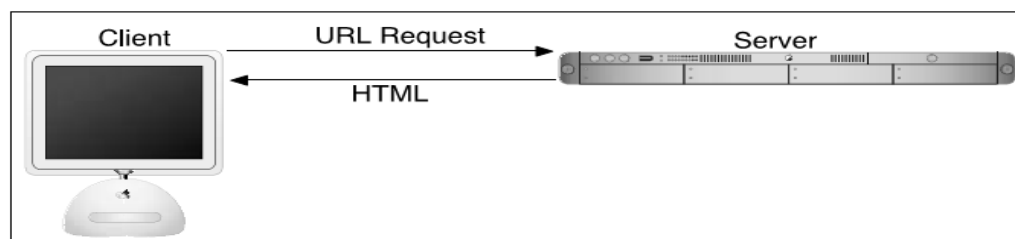


How pHP works?

As previously stated, PHP is a server-side language. This means that the code you write in PHP sits on a host computer called a server. The server sends Web pages to the requesting visitors (you, the client, with your Web browser). When a visitor goes to a Web site written in PHP, the server reads the PHP code and then processes it according to its scripted directions. In the example shown in below, the PHP code tells the server to send the appropriate data—HTML code—to the Web browser, which treats the received code as it would a standard HTML page. This differs from a static HTML site where, when a request is made, the server merely sends the HTML data to the Web browser and there is no server-side interpretation occurring. Because no server-side action is required, you can run HTML pages in your Web browser without using a server at all. To the end user and the Web browser there is no perceptible difference between what **home.html** and **home.php** may look like, but how that page's content was created will be significantly different.



D How PHP fits into the client/server model when a user requests a Web page.



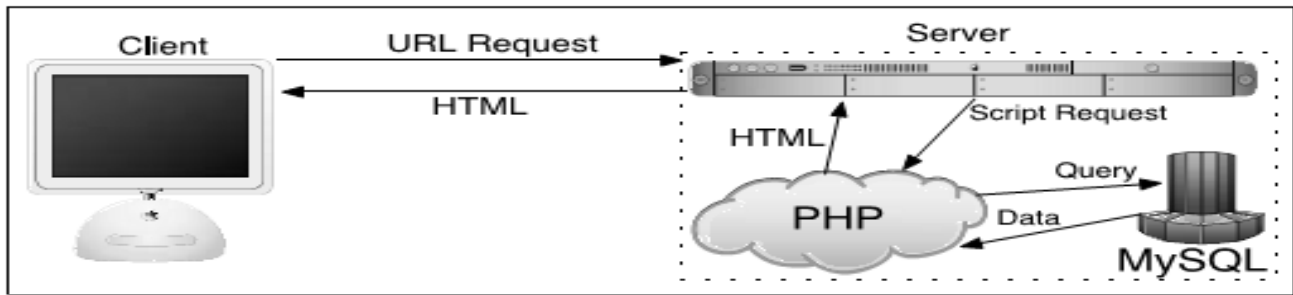
E The client/server process when a request for a static HTML page is made.

What is MySQL?

MySQL (www.mysql.com) is the world's most popular open-source database. In fact, today MySQL is a viable competitor to the pricey goliaths such as Oracle and Microsoft's SQL Server (and, ironically, MySQL is owned by Oracle). Like PHP, MySQL offers excellent performance, portability, and reliability, with a moderate learning curve and little to no cost.

MySQL is a database management system (DBMS) for relational databases (therefore, MySQL is an RDBMS). A database, in the simplest terms, is a collection of data, be it text, numbers, or binary files, stored and kept organized by the DBMS.

By incorporating a database into a Web application, some of the data generated by PHP can be retrieved from MySQL. This further moves the site's content from a static (hard-coded) basis to a flexible one, flexibility being the key to a dynamic Web site.



G How most of the dynamic Web applications in this book will work, using both PHP and MySQL.

MySQL has been known to handle databases as large as 60,000 tables with more than 5 billion rows. MySQL can work with tables as large as 8 million terabytes on some operating systems, generally a healthy 4 GB otherwise. MySQL is used by NASA and the United States Census Bureau, among many others.

Basic Syntax in PHP

PHP is an HTML-embedded scripting language, meaning that you can intermingle PHP and HTML code within the same file. So to begin programming with PHP, start with a simple Web page. Script 1.1 is an example of a no-frills, no-content XHTML Transitional document, which will be used as the foundation for most Web pages. Please also note that the template uses UTF-8 encoding, a topic discussed in the sidebar. To add PHP code to a page, place it with in PHP tags:

`<?php`

`?>`

Anything written within these tags will be treated by the Web server as PHP, meaning the PHP interpreter will process the code. Any text outside of the PHP tags is immediately sent to the Web browser as regular HTML. (Because PHP is most often

used to create content displayed in the Web browser, the PHP tags are normally put somewhere within the page's body.)

Along with placing PHP code within PHP tags, your PHP files must have a proper extension. The extension tells the server to treat the script in a special way, namely, as a PHP page. Most Web servers use **.html** for standard HTML pages and **.php** for PHP files.

To make a basic pHPscript:

1. Create a new document in your text editor or IDE, to be named first.php
2. Create a basic HTML document
3. Before the closing bodytag, insert the PHP tags:
`<?php`

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "http://www.w3.org/
  TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
3 <head>
4   <meta http-equiv="Content-Type"
     content="text/html; charset=utf-8" />
5   <title>Basic PHP Page</title>
6 </head>
7 <body>
8   <!-- Script 1.2 - first.php -->
9   <p>This is standard HTML.</p>
10 <?php
11 ?>
12 </body>
13 </html>
```

?>

4. Save the file as first.php.
5. Place the file in the proper directory of your Web server (localhost, webserver or IIS).
6. Run first.php in your Web browser (access them via a URL e.g. localhost/first.php).

Sending Data to the Web Browser

To create dynamic website with PHP, you must know how to send data to the Web browser. PHP has a number of built-in functions for this purpose, the most common being echo and print. I personally tend to favor echo: You could use print instead, if you prefer (the name more obviously indicates what it does):

echo 'Hello, world!';

echo "What's new?";

print 'Hello, world!';

print "What's new?";

What can PHP do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can restrict users to access some pages on your website
- PHP can encrypt data

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

Why PHP?

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

Writing Comments in PHP

<!DOCTYPE html>

<html>

<body>

<?php

//This is a PHP comment line

/*

This is a PHP comment

PHP Variables

Variables are "containers" for storing information:

\$x=5;

\$y=6;

\$z=\$x+\$y;

echo \$z;

?>

Much Like Algebra $x=5$, $y=6$, $z=x+y$

```
block

*/

?>

</body>

</html>
```

Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must begin with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name should not contain spaces Variable names are case sensitive (\$y and \$Y are two different variables)

Both PHP statements and PHP variables are case-sensitive.

Creating (Declaring) PHP Variables

PHP has no command for declaring a variable. A variable is created the moment you first assign a value to it:

```
$txt="Hello world!"; or $x=5;
```

After the execution of the statements above, the variable txt will hold the value Hello world!, and the variable x will hold the value 5.

Note: When we assign a text value to a variable, put quotes around the value.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In a strongly typed programming language, we will have to declare (define) the type and name of the variable before using it.

PHP Variable Scopes

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has four different variable scopes:

1. Local
2. global
3. static
4. parameter

1. Local Scope

A variable declared within a PHP function is local and can only be accessed within that function:

Example

```
<?php
```

The script above will not produce any output because the echo statement refers to the local scope variable \$x, which has not been assigned a value within this scope.

We can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

```

$x=5; // global scope

function myTest()
{
echo $x; // local scope
}

myTest();

?>

```

2. Global Scope

A variable that is defined outside of any function, has a global scope. Global variables can be accessed from any part of the script, EXCEPT from within a function.

To access a global variable from within a function, use the **global** keyword:

Example

```

<?php
$x=5; // global scope
$y=10; // global scope

function myTest()
{
global $x,$y;
$y=$x+$y;
}

myTest();

echo $y; // outputs 15

?> ** Note: how global key word is

```

3. Static Scope

When a function is completed, all of variable to not be deleted. To do this, use the static keyword when you first declare the variable:

Example

```

<?php

function myTest()

```

PHP also stores all global variables in an array called **\$GLOBALS[index]**.

The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

```

<?php
$x=5;
$y=10;

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

myTest();

echo $y;

?>

```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

4. Parameter Scope

A parameter is a local variable whose value is passed to the function by the calling code.

Parameters are declared in a parameter list as part of the function declaration:

```

{
static $x=0;

echo $x;

$x++;

}

myTest();

myTest();

myTest();

?>

```

PHP String Variables

A string variable is used to store and manipulate text.

String Variables in PHP

String variables are used for values that contain characters. After we have created a string variable we can manipulate it. A string can be used directly in a function or it can be stored in a variable. In the example below, we create a string variable called txt, then we assign the text "Hello world!" to it. Then we write the value of the txt variable to the output:

Example

```

<?php

$txt="Hello world!";

echo $txt;

?>

```

☺ Note: When we assign a text value to a variable, remember to put single or double quotes around the value.

The following are some commonly used functions and operators to manipulate strings.

The PHP Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to join two string values together. The example below shows how to concatenate two string variables together:

Example

```

<?php

$txt1="Hello world!";

$txt2="What a nice day!";

```

The output of the code above will be:

Hello world! What a nice day!

Tip: In the code above we have used the concatenation operator two times. This is because we wanted to insert a white space between the two strings.

```
echo $txt1 . " " . $txt2;
```

```
?>
```

The PHP strlen() function

Sometimes it is useful to know the length of a string value.

The strlen() function returns the length of a string, in characters. The example below returns the length of the string "Hello world!":

Example

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

The output of the code above will be: 12

Tip: strlen() is often used in loops or other functions, when it is important to know when a string ends. (i.e. in a loop, we might want to stop the loop after the last character in a string).

The PHP strpos () function

The strpos() function is used to search for a character or a specific text within a string. If a match is found, it will return the character position of the first match. If no match is found, it will return FALSE. The example below searches for the text "world" in the string "Hello world!":

Example

```
<?php
echo strpos("Hello world!","world");
?>
```

The output of the code above will be: 6

Tip: The position of the string "world" in the example above is 6. The reason that it is 6 (and not 7), is that the first character position in the string is 0, and not 1.

PHP Operators

The assignment operator = is used to assign values to variables in PHP.

The arithmetic operator + is used to add values together in PHP.

PHP Arithmetic Operators

Operator	Name	Description	Example	Result
x+y	Addition	Sum of x and y	2+2	4
x-y	Subtraction	Difference of x and y	5-2	3
x*y	Multiplication	Product of x and y	5*2	10
x/y	Division	Quotient of x and y	15/5	3
x%y	Modulus	Remainder of x divided by y	10 % 8	2
-x	Negation	Opposite of x	-2	2

a.b **Concatenation Concatenate two strings** **“hi”.”dear”** **hi dear**

PHP Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "\$x = 5" is 5.

Assignment	Same as...	Description
x=y	x=y	the left operand gets set to the value of the expression on the right
x += y	x=x+y	Addition
x -= y	x=x-y	Subtraction
x *= y	x=x*y	Multiplication
x /= y	x=x/y	Division
x %= y	x=x%y	Modulus
a .= b	a=a.b	Concatenate two strings

PHP Incrementing/Decrementing Operators

Operator	Name	Description
++ x	Pre-increment	Increments x by one, then returns x
x ++	Post-increment	Returns x, then increments x by one
-- x	Pre-decrement	Decrements x by one, then returns x
x --	Post-decrement	Returns x, then decrements x by one

PHP Comparison Operators

Comparison operators allows us to compare two values:

Operator	Name	Description	Example
x == y	Equal	True if x is equal to y	5==8 returns false
x === y	Identical	True if x is equal to y, they are of same type	5==="5" returns false
x != y	Not equal	True if x is not equal to y	5!=8 returns true
x <> y	Not equal	True if x is not equal to y	5<>8 returns true
x !== y	Not identical	True if x is not equal to y, or they are not of same type	5!== "5" returns true
x > y	Greater than	True if x is greater than y	5>8 returns false

<code>x < y</code>	Less than	True if x is less than y	<code>5 < 8</code> returns true
<code>x >= y</code>	Greater or equal	True if x is greater than or equal to y	<code>5 >= 8</code> returns false
<code>x <= y</code>	less than or equal	True if x is less than or equal to y	<code>5 <= 8</code> returns true

PHP Logical Operators

Operator	Name	Description	Example
<code>x and y</code> true	And	True if both x and y are true	<code>x=6,y=3 (x < 10 and y > 1)</code> returns true
<code>x or y</code>	Or	True if either or both x and y are true	<code>x=6,y=3, (x==6 or y==5)</code> returns true
<code>x xor y</code> false	Xor	True if either x or y is true, but not both	<code>x=6,y=3, (x==6 xor y==3)</code> returns false
<code>x && y</code> true	And	True if both x and y are true	<code>x=6,y=3, (x < 10 && y > 1)</code> returns true
<code>x y</code>	Or	True if either or both x and y are true	<code>x=6,y=3, (x==5 y==5)</code> returns false
<code>!x</code>	Not	True if x is not true	<code>x=6,y=3, !(x==y)</code> returns true

PHP Conditional Statements

Very often when we write code, we want to perform different actions for different decisions. We can use conditional statements in our code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...else if...else statement** - selects one of several blocks of code to be executed
- **switch statement** - selects one of many blocks of code to be executed

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>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<?php
$d=date("D");
if ($d=="Fri")
{
echo "Hello!<br />";
echo "Have a nice weekend!";
echo "See you on Monday!";
}
?>
```

The Else... If Statement

If you want to execute some code if one of several conditions are true 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!":

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

**** Note:** no space between else and if.

So in this we actually have 3 option to go based on the day today.

```

echo "Have a nice weekend!";

elseif ($d=="Sun")

echo "Have a nice Sunday!";

else

echo "Have a nice day!";

?>

```

PHP Switch Statement

The Switch statement in PHP is used to perform one of several different actions based on one of several different conditions. 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..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;
}

```

This is how it works:

- A single expression (most often a variable) is evaluated once
- The value of the expression is compared with the values for each case in structure
- If there is a match, the code associated with that case is executed
- After a code is executed, **break** is used to stop the code from running into the next case
- The default statement is used if none of the cases are true

Example

```

<?php
switch ($x)
{

```

```

    case 1:

```

If you assign \$x as 5 i.e \$x=5;

The output will be

No number between 1 and 3

```

    echo "Number 1";

    break;

    case 2:

    echo "Number 2";

    break;

    case 3:

    echo "Number 3";

    break;

    default:

    echo "No number between 1 and 3";

}

?>

```

Looping

Bounded loops versus unbounded loops

A **bounded** loop executes a fixed number of times—you can tell by looking at the code how many times the loop will iterate, and the language guarantees that it won't loop more times than that. An **unbounded** loop repeats until some condition becomes true (or false), and that condition is dependent on the action of the code within the loop. Bounded loops are predictable, whereas unbounded loops can be as tricky as you like.

PHP doesn't actually have any constructs specifically for bounded loops—**while**, **do-while**, and **for** are all unbounded constructs—but an unbounded loop can do anything a bounded loop can do.

While

The simplest PHP looping construct is while, which has the following syntax:

```
while (condition)
```

```
statement
```

The while loop evaluates the condition expression as a Boolean—if it is true, it executes statement and then starts again by evaluating condition. If the condition is false, the while loop terminates. Of course, just as with if, statement may be a single statement or it may be a brace-enclosed block. The body of a while loop may not execute even once, as in:

```

while (FALSE)

    print("This will never print.<BR>");

```

Or it may execute forever, as in this code snippet:

```
while (TRUE)
```

```
print("All work and no play makes Jack a dull boy.<BR>");
```

or it may execute a predictable number of times, as in:

```
$count = 1;
```

```
while ($count <= 10)
```

```
{
```

```
print("count is $count<BR>");
```

```
$count = $count + 1;
```

```
}
```

which will print exactly 10 lines.

Do-while

The do-while construct is similar to while, except that the test happens at the end of the loop. The syntax is:

```
do statement
```

```
while (expression);
```

The statement is executed once, and then the expression is evaluated. If the expression is true, the statement is repeated until the expression becomes false. The only practical difference between while and do-while is that the latter will always execute its statement at least once. For example:

```
$count = 45;
```

```
do
```

```
{
```

```
print("count is $count<BR>");
```

```
$count = $count + 1;
```

```
}
```

```
while ($count <= 10)
```

This prints the single line:

Output will be 45.

For Loop

The most complicated looping construct is for, which has the following syntax:

```
for (initial-expression; termination-check; loop-end-expression)
```

statement;

It is also legal to include more than one of each kind of for clause, separated by commas.

The termination-check will be considered to be true if any of its sub clauses are true; it is like an 'or' test. For example, the following statement:

```
for ($x = 1, $y = 1, $z = 1;    //initial expressions
    $y < 10, $z < 10;          // termination checks
    $x = $x + 1, $y = $y + 2,   // loop-end expressions
    $z = $z + 3)
    print("$x, $y, $z<BR>");
```

Would give the browser output:

1, 1, 1

2, 3, 4

3, 5, 7

Looping examples

create a Division Table using for loop.

<HTML>

<BODY>

<TABLE BORDER=1>

<?php

\$start_num = 1;

\$end_num = 10;

print("<TR>");

print("<TH> </TH>");

for (\$count_1 = \$start_num;

\$count_1 <= \$end_num;

\$count_1++)

print("<TH>\$count_1</TH>");

print("</TR>");

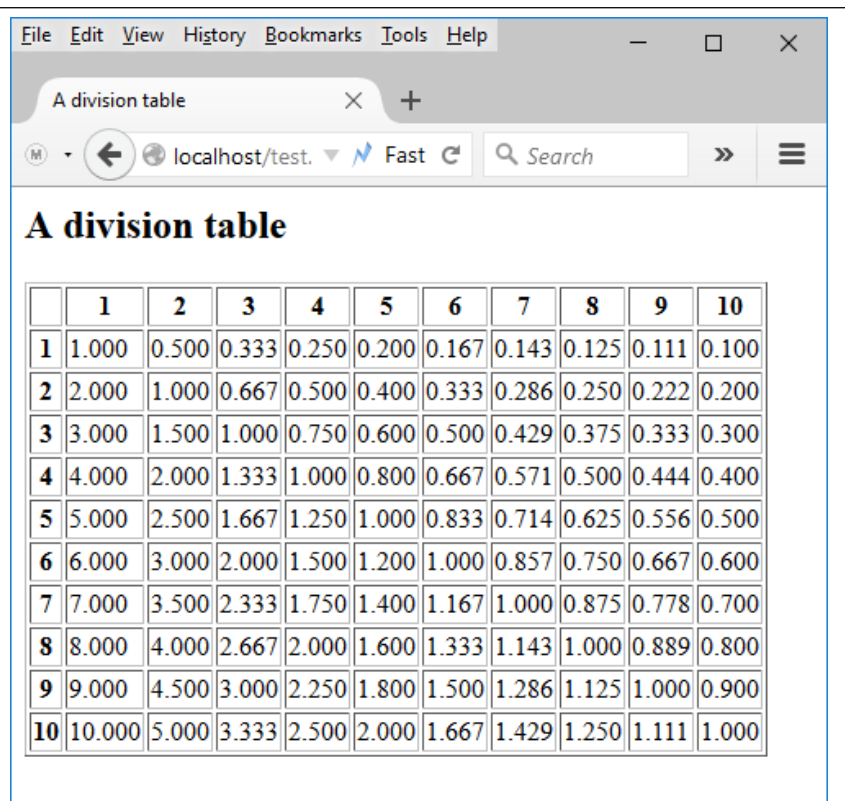
for (\$count_1 = \$start_num;

\$count_1 <= \$end_num;

\$count_1++)

{

print("<TR><TH>\$count_1</TH>");

A screenshot of a web browser window titled "A division table". The address bar shows "localhost/test." and the page content displays a table with 10 columns and 10 rows. The first row contains the numbers 1 through 10. The subsequent rows contain the results of dividing each number from 1 to 10 by the numbers in the first row. For example, the second row shows 1.000, 0.500, 0.333, etc. The table is styled with a border and alternating row colors.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 1.000 | 0.500 | 0.333 | 0.250 | 0.200 | 0.167 | 0.143 | 0.125 | 0.111 | 0.100 |
| 2 | 2.000 | 1.000 | 0.667 | 0.500 | 0.400 | 0.333 | 0.286 | 0.250 | 0.222 | 0.200 |
| 3 | 3.000 | 1.500 | 1.000 | 0.750 | 0.600 | 0.500 | 0.429 | 0.375 | 0.333 | 0.300 |
| 4 | 4.000 | 2.000 | 1.333 | 1.000 | 0.800 | 0.667 | 0.571 | 0.500 | 0.444 | 0.400 |
| 5 | 5.000 | 2.500 | 1.667 | 1.250 | 1.000 | 0.833 | 0.714 | 0.625 | 0.556 | 0.500 |
| 6 | 6.000 | 3.000 | 2.000 | 1.500 | 1.200 | 1.000 | 0.857 | 0.750 | 0.667 | 0.600 |
| 7 | 7.000 | 3.500 | 2.333 | 1.750 | 1.400 | 1.167 | 1.000 | 0.875 | 0.778 | 0.700 |
| 8 | 8.000 | 4.000 | 2.667 | 2.000 | 1.600 | 1.333 | 1.143 | 1.000 | 0.889 | 0.800 |
| 9 | 9.000 | 4.500 | 3.000 | 2.250 | 1.800 | 1.500 | 1.286 | 1.125 | 1.000 | 0.900 |
| 10 | 10.000 | 5.000 | 3.333 | 2.500 | 2.000 | 1.667 | 1.429 | 1.250 | 1.111 | 1.000 |

** The output would look like this.

```

for ($count_2 = $start_num;

$count_2 <= $end_num;

$count_2++)

{

$result = $count_1 / $count_2;

printf("<TD>%.3f</TD>", $result);

}

print("</TR>\n");

}

?>

</TABLE>

</BODY> </HTML>

```

PHP Arrays

An array can store one or more values in a single variable name.

What is an array?

When working with PHP, sooner or later, you might want to create many similar variables. Instead of having many similar variables, you can store the data as elements in an array. Each element in the array has its own ID so that it can be easily accessed. There are three different kind of arrays:

- Numeric array- An array with a numeric ID key
- Associative array- An array where each ID key is associated with a value
- Multidimensional array- An array containing one or more arrays

Numeric Arrays

A numeric array stores each element with a numeric ID key. There are different ways to create a numeric array.

Example 1

In this example the ID key is automatically assigned:

```
$names = array("Tsegaye","Abebe","Joe");
```

Example 2

In this example we assign the ID key manually:

```
$names[0] = "Tsegaye";
```

```
$names[1] = "Abebe";
```



```
$names[2] = "Joe";
```

The ID keys can be used in a script:

```
<?php  
  
$names[0] = "Tsegaye";  
  
$names[1] = "Abebe";  
  
$names[2] = "Joe";  
  
echo $names[1] . " and " . $names[2] . " are " . $names[0] . "'s neighbors";  
  
?>
```

Output: Abebe and Joe are Tsegaye's neighbors.

Associative Arrays

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
```

```
$ages['Quagmire'] = "30";
```

```
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php  
  
$ages['Peter'] = "32";  
  
$ages['Quagmire'] = "30";  
  
$ages['Joe'] = "34";  
  
echo "Peter is " . $ages['Peter'] . " years old."  
  
?>
```

The code above will output:

Peter is 32 years old.

Multidimensional Arrays

In a multidimensional 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.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
```

```
("Griffin"=>array("Peter", "Lois", "Megan"), "Quagmire"=>array("Glenn"), "Brown"=>array("Cleveland", "Loretta", "Junior"));
```

The array above would look like this if written to the output:

```
Array
```

```
([Griffin] => Array([0] => Peter[1] => Lois[2] => Megan) [Quagmire] => Array([0] => Glenn) [Brown] => Array([0] => Cleveland[1] => Loretta[2] => Junior) )
```

The foreach Statement

The foreach statement is used to loop through arrays.

For every loop, the value of the current array element is assigned to \$value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.

Syntax

```
foreach (array asvalue)
```

```
{
```

```
code to be executed;
```

```
}
```

Example

The following example demonstrates a loop that will print the values of the given array:

```
<?php
```

```
$arr=array("one", "two", "three");
```

```
foreach ($arr as $value)
```

```
{
```

```
echo "Value: " . $value . "<br />";
```

```
}
```

```
?>
```

PHP Functions

The real power of PHP comes from its functions. In PHP - there are more than 700 built-in functions available.

For a reference and examples of the built-in functions, please visit our [PHP Reference](#).

Create a PHP Function

A function is a block of code that can be executed whenever we need it.

Creating PHP functions:

- All functions start with the word "function()"
- Name the function - It should be possible to understand what the function does by its name. The name can start with a letter or underscore (not a number)
- Add a "{" - The function code starts after the opening curly brace
- Insert the function code
- Add a "}" - The function is finished by a closing curly brace.

Example

A simple function that writes my name when it is called:

```
<?php  
  
function writeMyName()  
{  
  
echo "Tsegaye Andargie";  
  
}  
  
writeMyName();  
  
?>
```

PHP Functions - Adding parameters

Our first function (writeMyName()) is a very simple function. It only writes a static string.

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

You may have noticed the parentheses after the function name, like: writeMyName(). The parameters are specified inside the parentheses.

Example 1

The following example will write different first names, but the same last name:

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

We can call a function as many times as we want.

The output of the program will be

My name is Tsegaye

My name is Andargie

```

}

echo "My name is ";

writeMyName("Tsegaye");

echo "My name is ";

writeMyName("Andargie");

?>

```

Example 2

The following function has two parameters:

```

<?php

function writeMyName($fname,$punctuation)

{

echo $fname . " Refsnes" . $punctuation . "<br />";

}

echo "My name is ";

writeMyName("Kai Jim", ".");

echo "My name is ";

writeMyName("Hege", "!");

echo "My name is ";

writeMyName("Ståle", "...");

?>

```

The output of the code above will be:

My name is Kai Jim Refsnes.

My name is Hege Refsnes!

My name is Ståle Refsnes...

PHP Functions - Return values

Functions can also be used to return values.

Example

```

<?php

function add($x,$y)

{

$total = $x + $y;

return $total;

```

The output of the code above will be:

1 + 16 = 17

```
}  
  
echo "1 + 16 = " . add(1,16)  
  
?>
```

Chapter Two- Forms

The PHP superglobals `$_GET` and `$_POST` are used to collect form-data. The example below displays a simple HTML form with two input fields and a submit button:

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

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

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

The output could be something like this:

```
Welcome John  
Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

```
<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>
```

and "welcome_get.php" looks like this:

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

PHP \$_REQUEST

PHP \$_REQUEST is used to collect data after submitting an HTML form using get or post method. e.g.

```
<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 = $_REQUEST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>
</body>
</html>
```

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as \$_GET and \$_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

\$_GET is an array of variables passed to the current script via the URL parameters.

\$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.



Developers prefer POST for sending form data.

PHP - Form Validation

Proper validation of form data is important to protect your form from hackers and spammers!

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male *

Your Input:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:

```
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male
```

The Form Element

The HTML code of the form looks like this:

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

When the form is submitted, the form data is sent with method="post".



What is the `$_SERVER["PHP_SELF"]` variable?

The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the `htmlspecialchars()` function?

The `htmlspecialchars()` function converts special characters to HTML entities. This means that it will replace HTML characters like `<` and `>` with `<` and `>`. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP `trim()` function)
2. Remove backslashes (`\`) from the user input data (with the PHP `stripslashes()` function)

We will name the function `test_input()`.

Now, we can check each `$_POST` variable with the `test_input()` function, and the script looks like this:

```
<?php  
// define variables and set to empty values  
$name = $email = $gender = $comment = $website = "";  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $name = test_input($_POST["name"]);  
    $email = test_input($_POST["email"]);  
    $website = test_input($_POST["website"]);  
    $comment = test_input($_POST["comment"]);  
    $gender = test_input($_POST["gender"]);  
}
```

```
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

In the following code we have added some new variables: \$nameErr, \$emailErr, \$genderErr, and \$websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each \$_POST variable. This checks if the \$_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
    }
    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = test_input($_POST["email"]);
    }
    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
    }
    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }
    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
?>
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
<label>Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">
</form>
```

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".



The `preg_match()` function searches a string for pattern, returning true if the pattern exists, and false otherwise.

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
        // check if name only contains letters and whitespace
        if (!preg_match("/^[a-zA-Z ]*$/", $name)) {

```

```

    $nameErr = "Only letters and white space allowed";
}
}

if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "Invalid email format";
    }
}

if (empty($_POST["website"])) {
    $website = "";
} else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows dashes in the URL)
    if (!preg_match("/^b(?:(:https?|ftp):\\|www\\.)[-a-z0-9+&@#\\/%?~_!:,.;]*[-a-z0-9+&@#\\/%?~_]/i", $website)) {
        $websiteErr = "Invalid URL";
    }
}

if (empty($_POST["comment"])) {
    $comment = "";
} else {
    $comment = test_input($_POST["comment"]);
}

if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
} else {
    $gender = test_input($_POST["gender"]);
}
}
?>

```

The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

Syntax

date(format,timestamp)

Parameter

Description

format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time



A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Simple Date

The required *format* parameter of the `date()` function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- `d` - Represents the day of the month (01 to 31)
- `m` - Represents a month (01 to 12)
- `Y` - Represents a year (in four digits)
- `l` (lowercase 'L') - Represents the day of the week

Other characters, like `"/`, `"."`, or `"-"` can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

PHP Tip - Automatic Copyright Year

Use the `date()` function to automatically update the copyright year on your website:

```
&copy; 2010-<?php echo date("Y")?>
```

Get a Simple Time

Here are some characters that are commonly used for times:

- `h` - 12-hour format of an hour with leading zeros (01 to 12)
- `i` - Minutes with leading zeros (00 to 59)
- `s` - Seconds with leading zeros (00 to 59)
- `a` - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

PHP - Include Files

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

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.

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

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of Framework, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

```
include 'filename';
or
require 'filename';
```

Example 1

Assume we have a standard footer file called "footer.php", that looks like this:

```
<?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>
```

Example 2

Assume we have a file called "vars.php", with some variables defined:

```
<?php
$color='red';
$car='BMW';
?>
```

Then, if we include the "vars.php" file, the variables can be used in the calling file:

Example

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

Chapter Three: Files and Directories

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!



When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language
```

Example

```
<?php  
echo readfile("webdictionary.txt");  
?>
```

The readfile() function is useful if all you want to do is open up a file and read its contents.

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

```
<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fread($myfile, filesize("webdictionary.txt"));  
fclose($myfile);  
?>
```

Tip: The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file

w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.



It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
    echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file. The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
    echo fgetc($myfile);
}
fclose($myfile);
?>
```

Note: After a call to the fgetc() function, the file pointer moves to the next character.

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

```
$myfile = fopen("testfile.txt", "w");
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string \$txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote will be present.

PHP - File Upload

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
```

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

```
<!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>
```

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.

Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
?>
```

PHP script explained:

- \$target_dir = "uploads/" - specifies the directory where the file is going to be placed
- \$target_file specifies the path of the file to be uploaded
- \$uploadOk=1 is not used yet (will be used later)
- \$imageFileType holds the file extension of the file
- Next, check if the image file is an actual image or a fake image



Note: You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and \$uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500kb, an error message is displayed, and \$uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting \$uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Complete Upload File PHP Script

The complete "upload.php" file now looks like this:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
    }
}
```

```

        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>

```

