Internet Programming II using PHP

PHP

PHP is a powerful server-side scripting language for creating dynamic and interactive websites.

PHP is the widely -used, free, and efficient alternative to competitors such as Microsoft's ASP. PHP is perfectly suited for Web development and can be embedded directly into the HTML code.

The PHP syntax is very similar to Perl and C. PHP is often used together with Apache (web server) on various operating systems. It also supports ISAPI and can be used with Microsoft's IIS on Windows.

Start learning PHP now!

Introduction to PHP

PHP is a server-side scripting language.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- Some scripting knowledge

If you want to study these subjects first, find the tutorials on our Home page.

What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

What is MySQL?

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

PHP + MySQL

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net

- PHP is easy to learn and runs efficiently on the server side

Where to Start?

To get access to a web server with PHP support, you can:

- Install Apache (or IIS) on your own server, install PHP, and MySQL
- Or find a web hosting plan with PHP and MySQL support

PHP Installation

What do You Need?

If your server supports PHP you don't need to do anything. Just create some .php files in your web directory, and the server will parse them for you. Because it is free, most web hosts offer PHP support.

However, if your server does not support PHP, you must install PHP.

Here is a link to a good tutorial from PHP.net on how to install PHP5:
http://www.php.net/manual/en/install.php

Download PHP

Download PHP for free here: http://www.php.net/downloads.php

Download MySQL Database

Download MySQL for free here: http://www.mysql.com/downloads/index.html

Download Apache Server

Download Apache for free here: http://httpd.apache.org/download.cgi

PHP Syntax

PHP code is executed on the server, and the plain HTML result is sent to the browser.

Basic PHP Syntax

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

For maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

```
<?php
?>
```

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

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

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

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

Note: The file must have the .php extension. If the file has a .html extension, the PHP code will not be executed.

Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>
<?php
//This is a comment
/*
This is
a commen
t block
*/
?>
</body>
</html>
```

PHP Variables

Variables are used for storing values, such as numbers, strings or function results, so that they can be used many times in a script.

Variables in PHP

Variables are used for storing a values, like text strings, numbers or

arrays. When a variable is set it can be used over and over again in your

script All variables in PHP start with a $ sign symbol. The correct way of

setting a variable in PHP:

```
$var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Let's try creating a variable with a string, and a variable with a number:

```
<?php
$txt = "Hello World!";
$number = 16;
?>
```

PHP is a Loosely Typed Language

In PHP a variable does not need to be declared before being set.

In the example above, you see that you do not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on how they are set.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP the variable is declared automatically when you use it.

Variable Naming Rules

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString)

PHP String

A string variable is used to store and manipulate a piece of text.

Strings in PHP

String variables are used for values that contains character strings.

In this tutorial we are going to look at some of the most common functions and operators used to manipulate strings in PHP.

After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the string "Hello World" to a string variable called $txt:

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

The output of the code above will be:

```
Hello World
```

Now, lets try to use some different functions and operators to manipulate our string.

The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to put two string values together.

To concatenate two variables together, use the dot (.) operator:

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

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

```
?>
```

The output of the code above will be:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

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

The output of the code above will be:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Complete PHP String Reference

For a complete reference of all string functions, go to our complete PHP String Reference.
The reference contains a brief description and examples of use for each function!

PHP Operators

Operators are used to operate on values.

PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y (gabungin jadi xy; bisa buat generate key)
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	And	x=6 y=3 (x < 10 && y > 1) returns true
||	Or	x=6 y=3 (x==5 || y==5) returns false
!	Not	x=6 y=3 !(x==y) returns true

PHP If...Else Statements

The if, elseif and else statements in PHP are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true

- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true

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:

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

```
</html>
```

The ElseIf 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!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>
</body>
</html>
```

PHP Switch Statement

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

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.
The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression)
{
case label1:
  code to be executed if expression =
  label1; break;
case label2:
  code to be executed if expression =
  label2; break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

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 the 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

```
<html>
<body>
<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
```

```
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>
</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("Peter","Quagmire","Joe");
```

Example 2

In this example we assign the ID key manually:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

The ID keys can be used in a script:

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

The code above will output:

```
Quagmire and Joe are Peter's neighbors
```

Associative Arrays

An associative array, each ID key is associated with a value.

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
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

PHP Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

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

The while Statement

The while statement will execute a block of code **if and as long as** a condition is true.

Syntax

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

Example

The following example demonstrates a loop that will continue to run as long as the variable i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br
  />"; $i++;
  }
?>
</body>
</html>
```

The do...while Statement

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

Syntax

```
do
{
code to be executed;
```

```
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<5);
?>
</body>
</html>
```

The for Statement

The for statement is the most advanced of the loops in PHP.

In it's simplest form, the for statement is used when you know how many times you want to execute a statement or a list of statements.

Syntax

```
for (init; cond; incr)
{
  code to be executed;
}
```

Parameters:

- **init**: Is mostly used to set a counter, but can be any code to be executed once at the beginning of the loop statement.
- **cond**: Is evaluated at beginning of each loop iteration. If the condition evaluates to TRUE, the loop continues and the code executes. If it evaluates to FALSE, the execution of the loop ends.
- **incr**: Is mostly used to increment a counter, but can be any code to be executed at the end of each loop.

Note: Each of the parameters can be empty or have multiple expressions separated by commas.

- **cond:** All expressions separated by a comma are evaluated but the result is taken from the last part. This parameter being empty means the loop should be run indefinitely. This is useful when using a conditional break statement inside the loop for ending the loop.

Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!<br />";
}
?>
</body>
</html>
```

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 as value)
{
    code to be executed;
}
```

Example

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

given array:

```
<html>
<body>
<?php
```

```
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
  echo "Value: " . $value . "<br />";
}
?>
</body>
</html>
```

PHP Functions

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

PHP Functions

In this tutorial we will show you how to create your own functions.
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:

```
<html>
<body>
<?php
function writeMyName()
```

```
  {
  echo "Kai Jim Refsnes";
  }
writeMyName();
?>
</body>
</html>
```

Use a PHP Function

Now we will use the function in a PHP script:

```
<html>
<body>
<?php
function writeMyName()
  {
  echo "Kai Jim Refsnes";
  }
echo "Hello world!<br />";
echo "My name is ";
writeMyName();
echo ".<br />That's right, ";
writeMyName();
echo " is my name.";
?>
</body>
</html>
```

The output of the code above will be:

```
Hello world!
My name is Kai Jim Refsnes.
That's right, Kai Jim Refsnes is my name.
```

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:

```
<html>
<body>
<?php
function writeMyName($fname)
  {
  echo $fname . " Refsnes.<br />";
  }
echo "My name is ";
writeMyName("Kai Jim");
echo "My name is ";
writeMyName("Hege");
echo "My name is ";
writeMyName("Stale");
?>
</body>
</html>
```

The output of the code above will be:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes.
My name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>
<body>
<?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","...");
?>
</body>
</html>
```

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

```
<html>
<body>
<?php
function add($x,$y)
  {
  $total = $x + $y;
  return $total;
  }
echo "1 + 16 = " . add(1,16);
?>
</body>
</html>
```

The output of the code above will be:

```
1+16=17
```

PHP Forms and User Input

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

Form example:

```
<html>
<body>
<form action="welcome.php"
method="post"> Name: <input type="text"
name="name" /> Age: <input type="text"
name="age" /> <input type="submit" />
</form>
</body>
</html>
```

The example HTML page above contains two input fields and a submit button. When the user fills in this form and click on the submit button, the form data is sent to the "welcome.php" file.

The "welcome.php" file looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years
old. </body>
</html>
```

A sample output of the above script may be:

```
Welcome John.
You are 28 years old.
```

The PHP $_GET and $_POST variables will be explained in the next chapters.

Form Validation

User input should be validated whenever possible. Client side validation is faster, and will reduce server load.

However, any site that gets enough traffic to worry about server resources, may also need to worry about site security. You should always use server side validation if the form accesses a database.

A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

PHP $_GET

The $_GET variable is used to collect values from a form with method="get".

The $_GET Variable

The $_GET variable is an array of variable names and values sent by the HTTP GET method.

The $_GET variable is used to collect values from a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and it has limits on the amount of information to send (max. 100 characters).

Example

```
<form action="welcome.php"
method="get"> Name: <input type="text"
name="name" /> Age: <input type="text"
name="age" /> <input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent could look something like this:

```
http://www.w3schools.com/welcome.php?name=Peter&age=37
```

The "welcome.php" file can now use the $_GET variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_GET array):

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

Why use $_GET?

Note: When using the $_GET variable all variable names and values are displayed in the URL. So this method should not be used when sending passwords or other sensitive information! However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note: The HTTP GET method is not suitable on large variable values; the value cannot exceed 100 characters.

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.

Example

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br /> You

are <?php echo $_REQUEST["age"]; ?> years old!
```

PHP $_POST

The $_POST variable is used to collect values from a form with method="post".

The $_POST Variable

The $_POST variable is an array of variable names and values sent by the HTTP POST method.

The $_POST variable is used to collect values from a form with method="post". Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Example

```
<form action="welcome.php" method="post">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age"
/> <input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will not contain any form data, and will look something like this:

```
http://www.w3schools.com/welcome.php
```

The "welcome.php" file can now use the $_POST variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_POST array):

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

are <?php echo $_POST["age"]; ?> years old!
```

Why use $_POST?

- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit

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

PHP Cookies

A cookie is often used to identify a user.

What is a 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.

How to Create a Cookie?

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

```
setcookie(name, value, expire, path, domain);
```

Example 1

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

```
<?php
setcookie("user", "Alex Porter",
time()+3600); ?>
<html>
.....
```

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "Alex Porter",
$expire); ?>
<html>
.....
```

In the example above the expiration time is set to a month (*60 sec * 60 min * 24 hours * 30 days*).

How to Retrieve a Cookie Value?

The PHP $_COOKIE variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
// Print a cookie echo
$_COOKIE["user"];
// A way to view all cookies
```

```
print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html>
<body>
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br
/>"; else
  echo "Welcome guest!<br />";
?>
</body>
</html>
```

How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

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

What if a Browser Does NOT Support Cookies?

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. One method is to pass the data through forms (forms and user input are described earlier in this tutorial).

The form below passes the user input to "welcome.php" when the user clicks on the "Submit" button:

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
```

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

Retrieve the values in the "welcome.php" file like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years
old. </body>
</html>
```

PHP Sessions

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

PHP Session Variables

When you are working 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 and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

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

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?php session_start(); ?>
<html>
<body>
</body>
</html>
```

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
session_start();
// store session data
$_SESSION['views']=1;
?>
<html>
<body>
<?php
//retrieve session data
echo "Pageviews=".
$_SESSION['views']; ?>
</body>
</html>
```

Output:

```
Pageviews=1
```

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

```
<?php
```

```
session_start();
if(isset($_SESSION['views']))
  $_SESSION['views']=$_SESSION['views']+1;

else
  $_SESSION['views']=1;
echo "Views=".
$_SESSION['views']; ?>
```

Destroying a Session

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

The unset() function is used to free the specified session variable:

```
<?php
unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

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

Note: session_destroy() will reset your session and you will lose all your stored session data.

PHP MySQL Introduction

MySQL is the most popular open-source database system.

What is MySQL?

MySQL is a database.

The data in MySQL is stored in database objects called tables.
A table is a collections of related data entries and it consists of columns and rows.
Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

Queries

A query is a question or a request.

With MySQL, we can query a database for specific information and have a recordset returned.

Look at the following query:

```
SELECT LastName FROM Persons
```

The query above selects all the data in the "LastName" column from the "Persons" table, and will return a recordset like this:

LastName
Hansen
Svendson
Pettersen

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download MySQL for free here: http://www.mysql.com/downloads/index.html

Facts About MySQL Database

One great thing about MySQL is that it can be scaled down to support embedded database applications. Perhaps it is because of this reputation that many people believe that MySQL can only handle small to medium-sized systems.

The truth is that MySQL is the de-facto standard database for web sites that support huge volumes of both data and end users (like Friendster, Yahoo, Google).

Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

PHP MySQL Connect to a Database

The free MySQL database is very often used with PHP.

Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database.

In PHP, this is done with the mysqli_connect() function.

Syntax

```
mysqli_connect(servername,username,password);
```

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important. Visit our full PHP MySQL Reference for more details.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
?>
```

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql_close() function:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
mysql_close($con)
; ?>
```

PHP MySQL Create Database and Tables

A database holds one or multiple tables.

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax

```
CREATE DATABASE database_name
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example creates a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }
mysql_close($con);
?>
```

Create a Table

The CREATE TABLE statement is used to create a table in MySQL.

Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

To learn more about SQL, please visit our SQL tutorial.

We must add the CREATE TABLE statement to the mysql_query() function to execute the command.

Example

The following example creates a table named "Persons", with three columns. The column names will be "FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }
// Create table
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE Persons
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";
// Execute query
mysql_query($sql,$con);
mysql_close($con);
?>
```

Important: A database must be selected before a table can be created. The database is selected with the mysql_select_db() function.

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MySQL, go to our complete Data Types reference.

Primary Keys and Auto Increment Fields

Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key

field cannot be null because the database engine requires a value to locate the record.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

Example

```
$sql = "CREATE TABLE Persons
(
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($sql,$con);
```

PHP MySQL Insert Into

The INSERT INTO statement is used to insert new records in a table.

Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to

be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statements above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
mysql_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Peter', 'Griffin', '35')");
mysql_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Glenn', 'Quagmire', '33')");
mysql_close($con);
?>
```

Insert Data From a Form Into a Database

Now we will create an HTML form that can be used to add new records to the "Persons" table.

Here is the HTML form:

```
<html>
<body>
<form action="insert.php" method="post"> Firstname:

<input type="text" name="firstname" />
```

```
Lastname: <input type="text" name="lastname" />
Age: <input type="text" name="age"
/> <input type="submit" />
</form>
</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php".

The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables.

Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table.

Here is the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')"
; if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";
mysql_close($con)
?>
```

PHP MySQL Select

The SELECT statement is used to select data from a database.

Select Data From a Database Table

The SELECT statement is used to select data from a database.

Syntax

```
SELECT column_name(s)
FROM table_name
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Persons" table (The * character selects all the data in the table):

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM Persons");
while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " .
  $row['LastName']; echo "<br />";
  }
mysql_close($con);
?>
```

The example above stores the data returned by the mysql_query() function in the $result variable.

Next, we use the mysql_fetch_array() function to return the first row from the recordset as an array. Each call to mysql_fetch_array() returns the next row in the recordset. The while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

```
Peter Griffin
Glenn Quagmire
```

Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo   "<td>"   .   $row['FirstName']   .
  "</td>"; echo "<td>" . $row['LastName'] .
  "</td>"; echo "</tr>";
  }
echo "</table>";
mysql_close($con);
?>
```

The output of the code above will be:

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

PHP MySQL The Where Clause

The WHERE clause is used to filter records.

The WHERE clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all rows from the "Persons" table where "FirstName='Peter':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM
Persons WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " .
  $row['LastName']; echo "<br />";
```

```
  }

?>
```

The output of the code above will be:

```
Peter Griffin
```

PHP MySQL Order By Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.
The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

To learn more about SQL, please visit our SQL tutorial.

Example

The following example selects all the data stored in the "Persons" table, and sorts the result by the "Age" column:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
```

```
mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons ORDER BY age");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br />";
  }
mysql_close($con);
?>
```

The output of the code above will be:

```
Glenn Quagmire 33
Peter Griffin 35
```

Order by Two Columns

It is also possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are equal:

```
SELECT column_name(s)
FROM table_name
ORDER BY column1, column2
```

PHP MySQL Update

The UPDATE statement is used to modify data in a table.

Update Data In a Database

The UPDATE statement is used to update existing records in a table.

Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Persons". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example updates some data in the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);

mysql_query("UPDATE Persons SET Age = '36'
WHERE FirstName = 'Peter' AND LastName =
'Griffin'"); mysql_close($con);
?>
```

After the update, the "Persons" table will look like this:

FirstName	LastName	Age
Peter	Griffin	36

Glenn	Quagmire	33

PHP MySQL Delete

The DELETE statement is used to delete records in a table.

Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

Syntax

```
DELETE FROM table_name
WHERE some_column = some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Look at the following "Persons" table:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example deletes all the records in the "Persons" table where LastName='Griffin':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);

mysql_query("DELETE FROM Persons WHERE
LastName='Griffin'"); mysql_close($con);
?>
```

After the deletion, the table will look like this:

FirstName	LastName	Age
Glenn	Quagmire	33

PHP Array Functions

PHP Array Introduction

The array functions allow you to manipulate arrays.
PHP supports both simple and multi-dimensional arrays. There are also specific functions for populating arrays from database queries.

Installation

The array functions are part of the PHP core. There is no installation needed to use these functions.

PHP Array Functions

PHP: indicates the earliest version of PHP that supports the function.

Function	Description	PHP
array()	Creates an array	3

array_change_key_case()	Returns an array with all keys in lowercase or uppercase	4
array_chunk()	Splits an array into chunks of arrays	4
array_combine()	Creates an array by using one array for keys and another for its values	5
array_count_values()	Returns an array with the number of occurrences for each value	4
array_diff()	Compares array values, and returns the differences	4
array_diff_assoc()	Compares array keys and values, and returns the differences	4
array_diff_key()	Compares array keys, and returns the differences	5
array_diff_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the differences	5
array_diff_ukey()	Compares array keys, with an additional user-made function check, and returns the differences	5
array_fill()	Fills an array with values	4
array_filter()	Filters elements of an array using a user-made function	4
array_flip()	Exchanges all keys with their associated values in an array	4
array_intersect()	Compares array values, and returns the matches	4
array_intersect_assoc()	Compares array keys and values, and returns the matches	4
array_intersect_key()	Compares array keys, and returns the matches	5
array_intersect_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the matches	5
array_intersect_ukey()	Compares array keys, with an additional user-made function check, and returns the matches	5

array_key_exists()	Checks if the specified key exists in the array	4
array_keys()	Returns all the keys of an array	4
array_map()	Sends each value of an array to a user-made function, which returns new values	4
array_merge()	Merges one or more arrays into one array	4
array_merge_recursive()	Merges one or more arrays into one array	4
array_multisort()	Sorts multiple or multi-dimensional arrays	4
array_pad()	Inserts a specified number of items, with a specified value, to an array	4
array_pop()	Deletes the last element of an array	4
array_product()	Calculates the product of the values in an array	5
array_push()	Inserts one or more elements to the end of an array	4
array_rand()	Returns one or more random keys from an array	4
array_reduce()	Returns an array as a string, using a user-defined function	4
array_reverse()	Returns an array in the reverse order	4
array_search()	Searches an array for a given value and returns the key	4
array_shift()	Removes the first element from an array, and returns the value of the removed element	4
array_slice()	Returns selected parts of an array	4
array_splice()	Removes and replaces specified elements of an array	4
array_sum()	Returns the sum of the values in an array	4
array_udiff()	Compares array values in a user-made function and returns an array	5
array_udiff_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_udiff_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_uintersect()	Compares array values in a user-made	5

	function and returns an array	
array_uintersect_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_uintersect_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_unique()	Removes duplicate values from an array	4
array_unshift()	Adds one or more elements to the beginning of an array	4
array_values()	Returns all the values of an array	4
array_walk()	Applies a user function to every member of an array	3
array_walk_recursive()	Applies a user function recursively to every member of an array	5
arsort()	Sorts an array in reverse order and maintain index association	3
asort()	Sorts an array and maintain index association	3
compact()	Create array containing variables and their values	4
count()	Counts elements in an array, or properties in an object	3
current()	Returns the current element in an array	3
each()	Returns the current key and value pair from an array	3
end()	Sets the internal pointer of an array to its last element	3
extract()	Imports variables into the current symbol table from an array	3
in_array()	Checks if a specified value exists in an array	4
key()	Fetches a key from an array	3
krsort()	Sorts an array by key in reverse order	3
ksort()	Sorts an array by key	3
list()	Assigns variables as if they were an array	3
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm	4

natsort()	Sorts an array using a "natural order" algorithm	4
next()	Advance the internal array pointer of an array	3
pos()	Alias of current()	3
prev()	Rewinds the internal array pointer	3
range()	Creates an array containing a range of elements	3
reset()	Sets the internal pointer of an array to its first element	3
rsort()	Sorts an array in reverse order	3
shuffle()	Shuffles an array	3
sizeof()	Alias of count()	3
sort()	Sorts an array	3
uasort()	Sorts an array with a user-defined function and maintain index association	3
uksort()	Sorts an array by keys using a user-defined function	3
usort()	Sorts an array by values using a user-defined function	3

PHP Array Constants

PHP: indicates the earliest version of PHP that supports the constant.

Constant	Description	PHP
CASE_LOWER	Used with array_change_key_case() to convert array keys to lower case	
CASE_UPPER	Used with array_change_key_case() to convert array keys to upper case	
SORT_ASC	Used with array_multisort() to sort in ascending order	
SORT_DESC	Used with array_multisort() to sort in descending order	
SORT_REGULAR	Used to compare items normally	
SORT_NUMERIC	Used to compare items numerically	