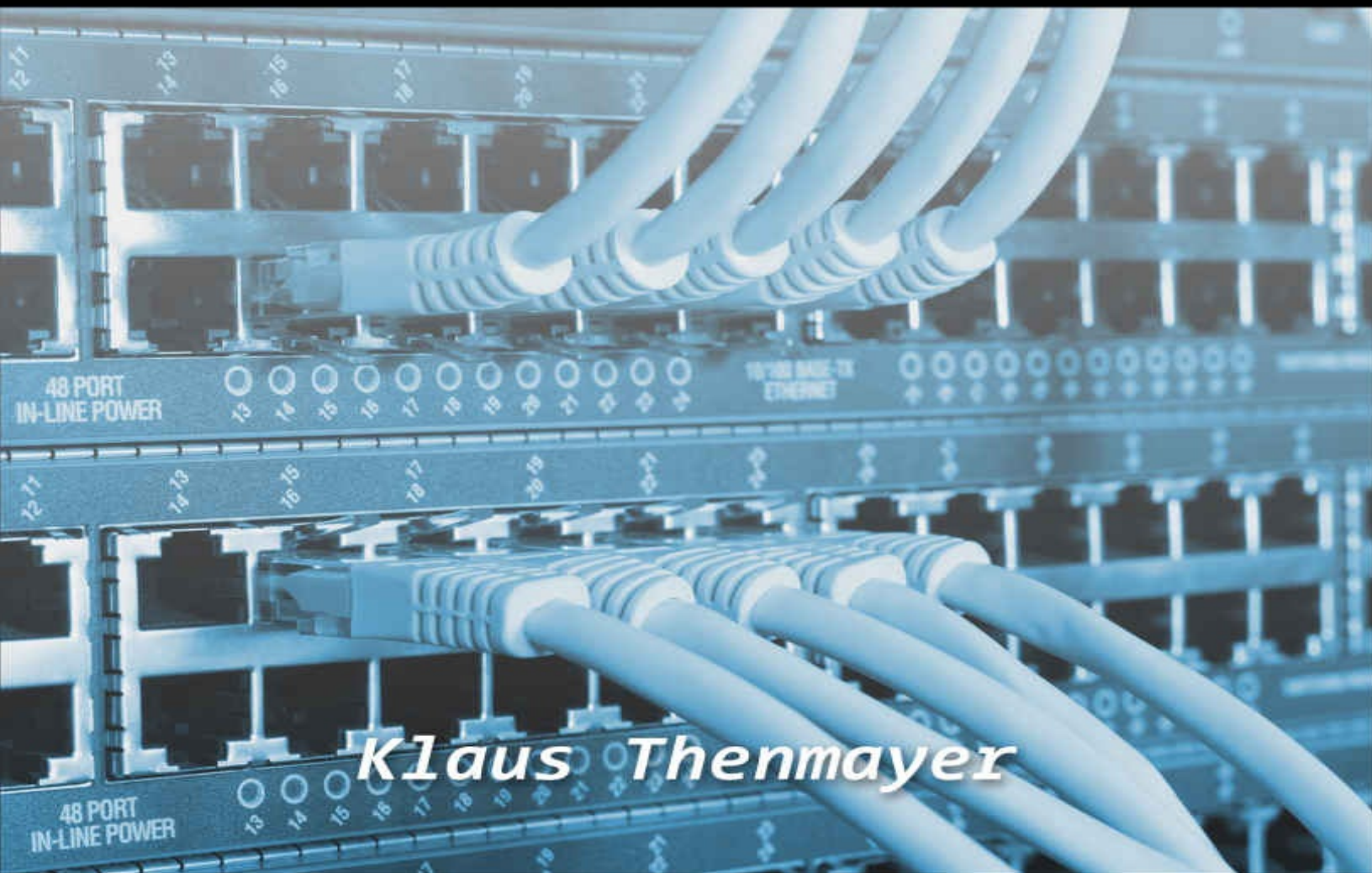
PHP Beginners Course

Understand the basics of PHP / MySQL programming in 5 days



Klaus Thenmayer

PHP Beginners Course
Understand basics of PHP / MySQL programming in 5 days

Imprint

Initial release May 2012 as ebook at Amazon KDP and Smashwords Inc.
English translation December 2016
6th edition January 2017

Cover design: Klaus Thenmayer
Cover Photo: © MWiner / Fotolia.com

Table of Content

Preamble

Welcome!

So you want to learn quickly how to program in PHP? This book is the right choice. You will learn within easy-to-understand chapters the basics of developing dynamic websites in PHP. It places special emphasis on full code examples which run independently of each other.

It begins with setting up the development environment on your computer. Immediately after you write your first PHP script and execute it in your newly installed server. Then you will learn how you can query information and control the program sequence in order to create dynamic websites. You also learn how you can save the information in databases or files, and read again.

Finally, you will get a short introduction to more advanced topics such as writing clean and safe program code.

You will not become a professional web developer by reading only this book, but you will get very good fundamentals and a guide in programming your first dynamic websites.

Prerequisites

There is no special knowledge necessary. If you want to learn programming in PHP, I assume that you know how to install applications on your computer.

Some knowledge of HTML to format texts cannot harm but is not mandatory and it will be the main HTML code explained in the appropriate places.

On hardware you need is a computer with monitor, mouse and keyboard. Since we program websites, of course an internet connection helps to get the needed software and more tutorials, but isn't needed during development, as we install our own little server. You only need to download some (free) software from the Internet, more on that later.

You are free, what operating system you use for development. The book will show you how to do the installation of a PHP server with MySQL support on Windows, Linux and MacOS and it will give you some examples of modern text editors.

Day 1

PHP – DEFINITION AND HISTORY

PHP is a scripting language that is used primarily to develop dynamic webpages. PHP scripts are executed directly on the server and not in the browser of a visitor.

PHP was developed in 1995 by Rasmus Lerdorf. His goal was to create a simple replacement for the collection of use-ful Perl scripts. This is how Personal Home Page Tools came into being. Soon after, he recreated the tool collection in the programming language C. Today, PHP is still developed in C, a very powerful programming language. It was soon followed by the first version, PHP / FI (Form Interpreter).

In 1998, Andi Gutmans and Zeev Suraski, in cooperation with Lerdorf, created PHP 3 from scratch to make it ready for use, particularly in the booming e-commerce business. Gutmans and Suraski founded the company Zend Technol-ogies Ltd., which contributed the main parts of PHP 4.

In 2004, PHP5 introduced enhanced support for object-oriented programming and greatly increased the speed of the PHP interpreter.

Eleven years later, in 2015, PHP 7 was released. It is powered by a new engine with a fast speed. PHP 6 was skipped because projects planned to carry out the program were never finished and so the developers focused on a completely new core engine. Most scripts execute twice as fast as in PHP 5.x. The majority of old PHP 5 scripts only need small changes to run in PHP 7. Scripts in this book are all updated to be fully compatible.

Setting up the development environment (Windows, Linux or Mac OS)

Windows

On Windows, the prefabricated server packages make it extremely easy to install the server. For the development, I prefer the package of Xampp (www.apachefriends.org).

Download the installer for Windows and run the installation. I recommend to keep C:/xampp/ as the default folder path. If you install the package under C:/Programs/XAMPP/, the server may make problems because the necessary file permissions are not set correctly. During the installation, it will be asked whether the individual server components should be installed as a service in the operating system. You must not do so, otherwise the server will constantly run in the background (until you stop the services manually), consume resources, and will be continuously available over the network.

Linux

In Linux, it would be better to install an Apache web server and a MySQL server using the Package Manager and to perform the configuration manually. But, for this book, I recommend that the beginners use a prepared package from www.apachefriends.org, which has all the necessary server features and PHP extensions. Beneath the download links, you will find the Linux installation instructions and some tips on how you can start and stop the server quickly.

MacOS

There are Apachefriends.org packages even for MacOS. But I prefer the package of MAMP for MacOS users. Its oper-ation is similar but has given me fewer problems. The application is available for free under www.mamp.info. There is also a paid Pro version, but we do not need this for our first programming experience. After downloading the setup file, please install MAMP in your Application folder.

All the abovementioned server packages contain a directory with the label ‘htdocs.’ We will use this to store files that can be accessed via a browser from the server. So, open your browser and enter the following URL: http://127.0.0.1

Under MacOS, you have to enter the port number of the server (the default number is 4444, so the entire URL should be: http://127.0.0.1:4444)

You should be greeted from your server package’s website (XAMPP / MAMP) and receive some information about the server.

127.0.0.1 always leads to the local computer. This means that the browser tries to find a local web server. Alternatively, you could use http://localhost. This should lead to the same result.

Text editors

Each operating system has a lot of good text editors. However, in this case, Microsoft Office Word or LibreOffice Writer cannot be regarded as text editors because they save a lot of information about the page and text formatting apart from the text. A text editor saves the exact characters of the entered text as it appears in the editor. Modern text editors colour text to make it more readable. The colouring will not be saved, but shown dynamically when you reopen the file. This function is called syntax highlighting. Syntax is the term used to describe the code, and highlighting it in a text editor means that various commands and functions of a programming language are coloured.

There are also Integrated Development Environments (IDEs), which help a lot to write and manage the program code and all other parts of a software project. IDEs automatically detect cross-references to functions in the program code and can display the recently used functions while writing notes.

There are large IDE projects, which are available as open source items, or under different licenses for all operating sys-tems. Many developers use the IDEs Eclipse (www.eclipse.org) or NetBeans (www.netbeans.org).

If you have a low-performance computer or want to quickly test some short PHP scripts, then I recommend using the text editor jEdit, which has been developed as an open source software and can be downloaded from www.jedit.org. In the larger Linux distributions, jEdit is available via a Software Channel (Package Manager) in a somewhat outdated version.

For this book, it does not matter whether you are using an IDE or a simple text editor. If you are going to be a profes-sional PHP developer, you will come in contact with an IDE sooner or later.

In order to issue a specific recommendation, I would personally suggest that you write the first scripts in a text editor like jEdit, and more advanced scripts into the surface of an IDE, only after finishing a handful of small PHP projects.

The first PHP program (Welcome the whole world)

We will now write our first small PHP script.

Open the text editor of your choice and type in the following lines of code:

```
<?php
 echo "Hello World!";
?>
```

Save this text in the htdocs folder inside the web server installation and name the file hello.php.

Already the file extension .php shows that there is a PHP script on one file. Our script begins with the line are PHP code and not 1 <?php, which our server will recognize that the following lines to ?>: 1 output, but to be first processed by our PHP interpreter.

In the world of programming, there are so-called interpreters and compilers. Compilers first process a complete program code and generate a finished program in machine language that can be supplied clean packed, for example in an .exe or .app file to the customer. This process is called compiling. The compilation of complex projects can sometimes take a long time. Especially in the field of dynamic web development the visitors don't want to wait to long for the site to be fully assembled. So PHP is always interpreted fresh out of the code, that is, on every page request the code line by line and went for quasi live assembled the page (it is still only the finished page delivered to the visitors).

In our short code example, we find also the line echo "Hello World!"; which is our original script. Each command within PHP is terminated with a semicolon, which is located at the end of the line. A command could be long over several lines, then only the last line is terminated with a semicolon. The echo command returns in PHP a text string to the visitor's browser. Strings must be enclosed by quotation marks, so that PHP knows what's a command and what's a string. Whether single or double quotes are used, in most cases, does not matter. Just be aware that once again includes all strings with the same character. Most text editors raise strings produce color and thus can be recognized, if they were again closed properly easy.

When we call now following in our browser the following address

http://127.0.0.1/hello.php (or http://127.0.0.1:4444/hello.php depending on the operating system)

our script from PHP gets executed.

If the server is set up correctly and the PHP script could be executed, then the following output is displayed within the browser:

```
Hello World!
```

If this is the case for you, congratulations on your first PHP script.

If not, it could be a possible source of error that either the server is not running or has disabled itself, or that the semicolon is missing at the end of the line of code with the echo command.

Variables (information stores)

One of the key components of a modern programming language are variables. These serve as small information stores. Variables can store values of various kinds. E.g. text characters (called String), numbers (without decimal digits they are named Integer, with Decimal or Float), a date or a number of other values.

In many programming languages you to define which type of values you want to save when you create a variable. So there are Char variables that can store text strings only, or so-called integer variables, where you can store numbers. PHP is dynamic in variable types. You don’t need to specify whether a variable can only store numbers, characters, or else a certain type of information. A variable in PHP can also first contain a text and later in the script only a sequence of numbers.

Later in this text you will learn how to anticipate or control the variable program runoff depending on the current value of a variable.

Initialize variables

In our previous first PHP script, we wrote the text "Hello World!". Now we would not set the text right behind the echo command, but save it first in a variable and then the echo command should give us the content of this variable.

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

In PHP variables are preceded by a dollar sign ($). The variable name is arbitrary, but it must not contain spaces, but only small and uppercase letters, numbers and underscores.

To print the contents of several variables in succession, they must be separated by a point:

```
<?php
  $part1 = "Hello ";
  $part2 = "World!";
  echo $part1 . $part2;
?>
```

This gives the same output as in our first example. We can also output text blocks with variables together:

```
<?php
 $part1 = "Hello";
 echo $part1 . " World!";
?>
```

If you are looking for errors in a program, use the command var_dump any time to find

out by means of the function, the nature of the information within a variable:

```
<?php
  $part1 = "Hello";
  var_dump($part1);
?>
```

Output:

```
string(5) "Hello"
```

This means that the contents of our variable is of type "string" and contains a string with 5 characters, namely "Hello".

Comment the code

If our scripts are only a few lines long, it is easy to recognize at first glance what a code block does. However, as soon as the script becomes longer, it may be necessary to leave comments within the program code, so that everyone who reads the code later immediately knows what the following lines of code object are.

There are single-line and multi-line comments in PHP. Line comments start with two forward slashes //. Anything written after the two slashes on the current line is no longer treated as a code.

Multi-line comments begin with a slash and a star, /*, and end with a star and a slash, */.

Here is another example:

```
<?php
  // I am a single line comment
  echo "I am code";
  /*
  I
  am
  a
  multiline
  comment
  */
?>
```

This script outputs only the following text:

```
I am code
```

A good programmer should, of course, always have as a goal writing a program code that reveals all purposes on its own. This can be achieved, for example, by logical variable and function names. Variables termed $a, $b, and $c indicate ‘less than’ if the variables are named $total_price, $sales_tax, and $gross_price.

Nevertheless, for example, it is helpful if you write meaningful comments when you are programming complex algorithms or codes. You could divide the program code via comments in several blocks.

So, should a program block start with the comment,

```
/* ********************
process user input
******************** */
```

and the next block is labelled as follows:

```
/* ********************
perform calculations
******************** */
```

and the last block, for example, is as follows:

```
/* ********************
output results
******************** */
```

then any programmer, who wants to update the calculations, knows immediately in which area of the code he has to look.

Arithmetic operations (basic arithmetic)

If you save only numbers in variables instead of letters, then you can calculate with these.

Here is an example (in this example, the HTML code
 is used, which is indicated in the browser as a line break):

```
<?php
  $number1 = 2;
  $number2 = 3;

  $addition = $number1 + $number2;
  $subtraction = $number1 - $number2;
  $multiplication = $number1 * $number2;
  $division = $number1 / $number2;

  echo "Number 1: " . $number1;
  echo "<br />";
  echo "Number 2: " . $number2;
  echo "<br />";
  echo "<br />";

  echo "Addition: " . $addition;
  echo "<br />";
  echo "Subtraction: " . $subtraction;
  echo "<br />";
  echo "Multiplication: " . $multiplication;
  echo "<br />";
  echo "Division: " . $division;
?>
```

This script has the following output in the browser:

```
Number 1: 2
Number 2: 3

Addition: 5
Subtraction: -1
Multiplication: 6
Division: 0.66666666666667
```

There is an alternative spelling for simple arithmetic:

```
<?php
  $value1 = 2;
  $value2 = 3;
  $value1 = $value1 + $value2;
  $value2 = $value2 * $value2;

  echo $value1;
  echo "<br />";
  echo $value2;
?>
```

It gives the same output as the following:

```
<?php
  $value1 = 2;
  $value2 = 3;
  $value1 += $value2;
  $value2 *= $value2;

  echo $value1;
  echo "<br />";
  echo $value2;
?>
```

Output:

```
5
9
```

As a small task, you could write a script that calculates the following:

One website has 30,000 visitors daily.

An ad-blocking plug-in is used by 60% of the visitors. In other words, 40% of the visitors see a banner advertisement on our website. Of these 40% visitors, 10% click on the banners. For every 1,000 clicks, we get €200.

How much do we earn per month (calculated as 30 days)?

Use all values in the variables; then you can calculate very fast how much we could earn when our website has 100,000 visitors daily.

The answer is on the next page.

As a solution, you should get the following: €240 and €800.

Example solution:

```
<?php
  $dailyVisitors = 30000;
  $seesads = 0.4; // 60% get blocked
  $clickratio = 0.1;
  $earnings = 200; // per 1000 clicks

  $ads = $dailyVisitors * $seesads;
  $clicks = $ads * $clickratio;
  echo ($clicks / 1000) * $earnings;
?>
```

Combining several variables (arrays)

If you have several variables to represent a number of similar values, it makes sense to include them in an array.

Here are two examples that have the same output:

```
<?php
  $value1 = 2;
  $value2 = 4;
  $value3 = 8;
  $value4 = 16;
  $value5 = 32;

  echo $value1 . "<br />";
  echo $value2 . "<br />";
  echo $value3 . "<br />";
  echo $value4 . "<br />";
  echo $value5 . "<br />";
?>
```

```
<?php
  $valueArray = array(2,4,8,16,32);

  echo $valueArray[0] . "<br />";
  echo $valueArray[1] . "<br />";
  echo $valueArray[2] . "<br />";
  echo $valueArray[3] . "<br />";
  echo $valueArray[4] . "<br />";
?>
```

Both have the following output:

```
2
4
8
16
32
```

You can access the values of an array by using $array[index]. The entries are numbered in ascending order from 0 to the number of elements. You can also set the index by yourself:

```
<?php
  $myArray = array("index1" => "value1");
  echo $myArray["index1"];

  echo "<br />";

  $myArray["index1"] = "value2";
  echo $myArray["index1"];
?>
```

Output:

```
value1
```

```
value2
```

Later, we will learn how to automate the output of all the values in an array in succession and process them. This is useful in arrays that contain dozens of entries.

Sort arrays

If we fill an array dynamically and we want to sort the values alphabetically, there is a simple function: sort();

```
<?php
  $myArray = array("Zita", "Anton", "Hans");

  // unsorted output
  echo $myArray[0];
  echo "<br />";
  echo $myArray[1];
  echo "<br />";
  echo $myArray[2];

  // Sort array
  sort($myArray);
  echo "<br /><br />";

  // Sorted output
  echo $myArray[0];
  echo "<br />";
  echo $myArray[1];
  echo "<br />";
  echo $myArray[2];
?>
```

Output:

```
Zita
Anton
Hans

Anton
Hans
Zita
```

Multidimensional arrays

An array could, again, contain as a value, another array:

```
<?php
  $myArray = array();
  $myArray["index1"] = array("value1", "value2");
  $myArray["index2"] = array("value3", "value4");

  echo $myArray["index1"][0];
  echo "<br />";
  echo $myArray["index2"][1];
?>
```

Output:

```
value1
value4
```

So, you could save coordinates and read them, for example, as a coordinate system with X, Y:

```
<?php
  [...]
  echo $chessboard[10][2];
?>
```

This would depend on how your script is constructed. For example, you could show the pitch in the 10th row and the second column.

Day 2

Fill variables with user information (GET / POST)

So far, although we have had our variables always filled with values within the code, the result has not been a dynamic website. We now want to receive information from ‘outside’.

For a user to send to our PHP script data, there are, among other things, two simple methods: POST and GET.

POST information is transmitted, for example, in an appropriately configured form to our server when the user is registered for or, when he makes a new forum entry. Then, there is the GET information, which is received via invoked URL. You must have noticed the long URLs in web shops, which include many letters and numbers and automatically generated parts.

Identify and read GET variables

GET variables are passed as parameters in an URL:
http://www.myurl.com/script.php?variable1=value1&variable2=value2&variable3=value3

The number of variables and values is virtually not limited, but old browsers and a lot of servers have a limit of 1024 characters in the URL.

When this URL is called in the browser, the computer first attempts to address the call, www.myurl.at. The file is then requested from the server, script.php, and three variables are received. The first variable has a question mark (?), the other variables, an ampersand (&).

Let us expand our welcome script to a dynamic component to greet our visitors in my name:

```
<?php
  $vistorsname = $_GET['name'];
  echo "Hello ".$vistorsname;
?>
```

Save the script, for example, under the name hello2.php in your htdocs folder.

Within PHP, there are always some automatically existing variables. There are, for example, an array called $_GET and one named $_POST from which the transmitted GET and POST information can be read. These variables are arrays, and the name of the values are the array index.

If you now call the script by the URL,
http://127.0.0.1/hello2.php?name=Klaus
the following text appears in the browser:

```
Hello Klaus
```

Congratulations! You have, now, programmed your first dynamic website and you are a bit ahead of other programmers who can write only simple HTML.

Identify and read POST variables

Would you query longer texts, including special characters, so that you could get on with the transmission via GET variables in the URL problems? For such purposes, there is the POST method, which is used as follows:

my_form.html

```
<form method="POST" action="read_form.php">
  <textarea name="myText"></textarea>
  <input type="submit" value="Send" />
</form>
```

read_form.php

```
<?php
  $myText = $_POST["myText"];
  echo nl2br($myText);
?>
```

In the first file, we define a form by HTML. Each form element from which we want to get information, gets the property (tag attributes) name='...'. This enables us to extract all information through the global variable, $_POST, in our second file.

Since we use a text box with multiple lines, we have, on the issue, in our PHP script the function nl2br, which stands for a new line to break or to make line breaks in the text output automatically
.

Conditions, program control (IF/ELSE/ELSEIF)

We can, already, according to the user input, output dynamic text, but we cannot run our script differently depending on user input. In programming, there are many cases, which can be written as follows: when the user, A, enters exactly something, do this, otherwise, do that. In PHP, this is written in English and a few diacritical marks.

```
if ($input == "A") {
  do this;
}
else {
  do that;
}
```

'If', followed by a condition written within normal parentheses ( ), the following code is only executed when the condition in the parentheses ( ) is met. The code block after the parentheses should be writtin within two brackets { }.

If the conditions inside the parentheses () are not applied, the else block, also bounded by braces, will be executed. Then the program will continue underneath the braces with the rest of the script.

One can combine 'if' and 'else' also to successively query multiple conditions and even attach an 'else' clause if none of the previous conditions are true:

```
if ($input == "A") {
  do this;
}
elseif ($input == "A") {
  do this;
}
else {
  do that;
}
```

Conditions are compared with different characters. If a variable has a certain value, you have to write two actual equal signs ==. If you write just one equal sign, =, the program will simply store the value on the right side of the quotation mark in the variable.

Other conditions are defined as follows:
== equals
!= is not equal
> is greater
< less
>= greater than or equal
<= less than or equal

Several conditions can be linked with a double ampersand, &&. A double vertical line

would mean an OR operation ||.

Thus, $a == 1 && $b == 1 would mean that both the variables, $a and $b, must contain the value 1. If one variable had another value, the else clause would be executed.

$a == 1 || $b == 1 would mean that either the variable, $a or the variable, $b or both the variables must contain the value 1. You can combine multiple conditions by putting them in further parantheses:

if (($a == 1 && $b == 1) || $c == 1) would mean that either $a and $b must have the value 1 or $c must have the value 1.

To make the benefits of the if-else function more understandable, here is an example:

```
<?php
  $username = $_GET["username"];
  $password = $_GET["password"];

  if ($username == "admin" && $password == "mystrongandsecretpassword") {
    echo "Welcome Administrator!";
  }
  elseif ($username == "admin" && $password != "mystrongandsecretpassword") {
    echo "Hello administrator, unfortunately, you have entered the password
incorrectly!";
  }
  else {
    echo "Error: The login data entered were incorrect!";
  }
?>
```

In the script above, first, the values from the GET array are stored in variables. Of course, it would also be possible to use the GET array directly in the if-function, but that would make the code more complicated.

If a visitor in the URL has transmitted the correct login information, we welcome him as an administrator, otherwise, we will check if he has, at least, entered the user name correctly. If the username and password are incorrect, we will give an error message.

Output when calling
script.php?username=admin&password=mystrongandsecretpassword

```
Welcome Administrator!
```

Output when calling
script.php?username=admin&password=xxx

```
Hello administrator, unfortunately, you have entered the password incorrectly!
```

Output when calling
script.php?username=abc&password=xxx

```
Error: The login data entered were incorrect!
```

If the contents of a variable do not matter, and we just want to check if a variable has any content, then we can accomplish this as follows:

```
<?php
  $myTrueValue = "myTest";
  if ($myTrueValue) {
    echo 1;
  }
  else {
    echo 0;
  }

  echo "<br />";

  $myTrueValue = "";
  if ($myTrueValue) {
    echo 1;
  }
  else {
    echo 0;
  }
?>
```

Output:

```
1
0
```

In the first query, if the variable has a content, then the ‘if’ block and not the ‘else’ block is executed. In the second review, the variable is empty. When you do not test any condition like == or !=, ..., it is checked whether the variable is empty or contains a Boolean value. A Boolean value consists in programming as a variable or a value of true or false, it can also be written as 1 or 0.

Here, the previous example is slightly extended:

```
<?php
  $myTrueValue = "myTest";
  if ($myTrueValue) {
    echo 1;
  }
  else {
    echo 0;
  }

  echo "<br />";

  $myTrueValue = false;
  if ($myTrueValue) {
    echo 1;
  }
  else {
    echo 0;
```

```
  }

  echo "<br />";

  $myTrueValue = 0;
  if ($myTrueValue) {
    echo 1;
  }
  else {
    echo 0;
  }
?>
```

Output:

```
1
0
0
```

Furthermore, it is often important to know whether a variable exists at all and is, therefore, initialised in the program already:

```
<?php
  $myVar = "123";
  if (isset($myVar)) {
    echo "myVar exists";
  }
  else {
    echo "myVar does not exist";
  }
?>
```

Output:

```
myVar exists
```

```
<?php
  $myVar = "123";
  if (isset($otherVar)) {
    echo "myVar exists";
  }
  else {
    echo "myVar does not exists";
  }
?>
```

Output:

```
otherVar does not exists
```

Case analysis (switch / case)

Let us suppose we have a variable and want to check for many different contents. We already know the function of if, elseif and else. However, it may quickly become confusing; moreover, it is easier to write a redundant code to check only a single variable. There is the practical function switch case which is programmed as follows:

```
<?php
  $username = $_GET["username"];

  switch ($username) {
    case "Klaus":
      echo "Hello administrator!";
      break;
    case "Dagmar":
      echo "Hello sister!";
      break;
    case "Bill Gates":
      echo "Hello richest man in the world!";
      break;
    default:
      echo "Hello visitor!";
      break;
  }
?>
```

As we see, the head of the switch function is pretty similar to the ‘if’ function; it just has no parameters defined. Thereafter, each multiple case block must be completed with a break; that breaks the review on to other cases.

If, in neither case, the condition is met, the default block is executed. If you do not use breaks, you could run the same for two or more case code blocks:

```
<?php
  $username = $_GET["username"];

  switch ($username) {
    case "Klaus":
    case "Dagmar":
      echo "Hello administrator!";
      break;
    case "Bill Gates":
      echo "Hello richest man in the world!";
      break;
    default:
      echo "Hello visitor!";
      break;
  }
?>
```

Loops for repetitive tasks (While / For / Foreach)

While() Loop

Often, there are tasks for our script to be repeated many times. This could be, for example, a kind of numbering in a particular order:

```
<?php
  $number = 1;
  while ($number < 100) {
    echo $number . "<br />";
    $number++;
  }
?>
```

This short script gives back each number 1 to 99. It is used in a so-called while() loop. The header of the while() loop is used to specify a condition as an ‘if’ function. The loop is repeated as long as the condition is true or till a break, a command is invoked inside the loop.

The script also has a small gimmick. It took over from C++ to PHP: ++ increases the value of a variable by +1, -- reduces it to -1.

Here is another example of the same output, only this time, it is written in a quasi-infinite loop because the condition is true. The loop runs until there is a break; the command is cancelled:

```
<?php
  $number = 1;
  while (true) {
    echo $number . "<br />";
    $number++;
    if ($number > 99) {
      break;
    }
  }
?>
```

For() Loop

For fixed numbers, there is a much more beautiful loop: the for() loop.

The function of the for() loop is as follows:

for (start value; condition; to repeat command at each execution) {
}

Here is a brief example:

```
<?php
  for ($x = 1; $x < 100; $x++) {
```

```
    echo $x . "<br />";
  }
?>
```

This would result in the same output as we had with our while() loop. The third parameter of the function specifies a command to be executed after each loop. In this case, we increase the value of the variable $x by 1. The variable can be used only within the loop.

Foreach() loop

There is another loop function to work through arrays: foreach()

Here is an example of how this is used:

```
<?php
  $valueArray = array(2,4,8,16,32);
  foreach ($valueArray as $index => $value) {
    echo $value . "<br />";
  }
?>
```

Output:

```
2
4
8
16
32
```

Within the function, the $index and $value variables are available. With $index, you can determine which entry you have entered within the array (counted from 0 ascending), or if you have defined the index of an array entry, you have this within the loop. The names of the variables are freely definable; but it should be clear within the loop code block what the variables are used for.

If you do not need the array index within the loop, for example, as you only want to output the values in sequence, it is even easier:

```
<?php
  $valueArray = array(2,4,8,16,32);

  foreach ($valueArray as $value) {
    echo $value . "<br />";
  }
?>
```

Day 3

Date and Time (Who has turned the clock?)

DATE()

With the function date(format [, time]), you can print a certain time in a formatted form. The first parameter specifies the format, while a second optional parameter states a specific time. If the second parameter is absent, the current server time is as shown below.

Here is an example:

```
<?php
  echo date("Y-m-d");
?>
```

Output (if the actual server time is 24 December 2016):
2016-12-24

The second parameter must be specified in the PHP time() format:

This output would be the same:

```
<?php
  echo date("Y-m-d", time());
?>
```

The time() function returns the desired time as a number. The value is always the number of seconds after the UNIX time. This began on 1 January 1970 through 00:00:00 GMT.

microtime()

More precisely, but not suitable for the date() function, is the function of microtime(). The parameter can specify optionally whether you want to receive the return value as point number. The function returns the current date in the UNIX time in the exact number of microseconds. This is, for example, useful for measuring how long a script has run in order to find programs that would still have potential for optimization:

```
<?php
  $start = microtime(true);

  /*
  Two dummy variables, so that we can generate some computing time for the
script:
  */
  $dummyVariable1 = 1;
  $dummyVariable2 = 3;

  $x = 0;
  while ($x < 1000000) {
    $x++;

    $dummyVariable1 *= $dummyVariable2;
    $dummyVariable1 = $dummyVariable1 / $dummyVariable2;
  }
```

```
  $end = microtime(true);
  echo "Runtime: " . ($end - $start) . "ms";
?>
```

This has, for example, the output:

```
Runtime: 0.0886830329895ms
```

Give a point in time to DATE() function with mktime()

If you want to specify a certain time, you have to do this, for example, with the function mktime(hour, minute, second, month, day, year):

```
<?php
  echo date("Y-m-d", mktime(0, 0, 0, 12, 24, 2012));
?>
```

It would have the same output again.

strtotime()

To forestall a function from the chapter of text functions from this book, I shall tell you the right strtotime("date in text format"). With this, you can convert one written text in human readable form of a certain time in the time() format for the date() function:

```
<?php
  echo date("Y-m-d", strtotime("May 17th, 2012"));
?>
```

Some examples of such conversions are as follows:

```
strtotime("now")
strtotime("2012-05-17")
```

You can even do some simple calculations from the actual point in time:

```
strtotime("next Friday")
strtotime("last Monday")
strtotime("+1 day")
strtotime("+1 week 2 days 3 hours 4 seconds")
```

The following formats are possible:

Day

d

Day of the month, two digits with leading zeros

01 to 31

D

Weekday, truncated to three letters

Mon to Sun

j
Day of the month without leading zeros
1 to 31

l (lower case 'L')
Full weekday
Sunday to Saturday

N
Numeric representation of the day of the week according to ISO 8601 (in PHP 5.1.0 added)
1 (Monday) to 7 (Sunday)

S
English ordinal suffix for day of the month, two characters. Recommended for use with j
st, nd, rd or th.

w
Numeric day of the week
0 (Sunday) to 6 (Saturday)

z
The day of the year (starting from 0)
0 to 365

Week
W

ISO 8601 week number of year, weeks starting on Monday
Example: 42 (the 42nd week of the year)

Month
F
Month as a whole word, such as January or March
January to December

m
Month as a number, with leading zeros
01 to 12

M
Month name with three letters
Jan to Dec

n
Month number without leading zeros
1 to 12

t
Number of days of that month
28 to 31

Year
L
Leap year or not
1 is when it is a leap year, otherwise 0

o
Year, in accordance with ISO8601. This is the same value as Y, except when the ISO calendar week (W) belongs to the previous or next year, in which case that year is used.
Example: 1999 or 2003

Y
Four-digit year
Example: 1999 or 2003

y
Year, double-digit
Example: 99 or 03

Time
a
Lowercase ante meridiem (morning) and post meridiem (afternoon)
am or pm

A
Capital letters: ante meridiem (morning) and post meridiem (afternoon)
AM or PM

B
Swatch Internet Time
From 000 to 999 per day

g
Hour in 12-hour format, without leading zeros
1 to 12

G

Hour in 24-hour format, without leading zeros
0 to 23

h
Hour in 12-hour format, with leading zeros
01 to 12

H
Hour in 24-hour format, with leading zeros
00 to 23

i
Minutes, with leading zeros
00 to 59

s
Seconds, with leading zeros
00 to 59

u
Microseconds
Example: 654321

Time zone
e
Name of the time zone
Example: UTC, GMT, Atlantic/Azores

I (upper case 'i')
A date in the daylight (summer) time
1 if it is in daylight time, otherwise 0.

O
Difference to Greenwich time (GMT) in hours
Example: +0200

P
Difference to Greenwich time (GMT) in hours with colon between hours and minutes
Example: +02:00

T
Abbreviation for time zone
Example: EST, MDT

Z

Time zone offset in seconds. The offset for time zones west of UTC is always negative and for time zones east of UTC it is always positive.

-43200 to 50400

Complete date/time

c

ISO 8601 date

2004-02-12T15:19:21+00:00

r

According to RFC 2822 formatted date

Example: Thu, 21 Dec 2000 16:01:07 +0200

U

Seconds since the Unix epoch (1 January 1970, 00:00:00 GMT)

Text functions (String operations)

In PHP, there are a large number of functions for manipulating text, such as splitting a longer string and assemble it different again, search for characters, and more. I wish to show you the key functions, so that you can solve almost all the problems of the manipulation of strings. There are many other functions, which would be beyond the scope of this book for beginners. But you can find some links at the end of the book to sites where all available PHP functions are explained.

str_replace

A function that is commonly used is str_replace. This function looks at a text after the occurrence of a particular string and replaces it. The first parameter is the text to replace, the second is the one to be used instead of this text, and the third is the text to be searched in. The function returns the complete text with the replaced needles. As an optional parameter, you could specify the number of replacements to be performed.

Here is an example for application:

```
<?php
  $myText = "I am learning to program Java";
  echo $myText;
  $myText = str_replace("Java", "PHP", $myText);
  echo "<br />";
  echo $myText;
?>
```

Output:

```
I am learning to program Java
I am learning to program PHP
```

substr

Another text function that is used frequently is substr(). This is the abbreviation for substring. It allows you to detach a part of a string. The first parameter that the function expects is the string, the second is the point – counting from 0, to get returned for the first character from the part of your text (substring) – and as an optional parameter, the third one is the length of the returned string.

Here is another example:

```
<?php
  $myText = "I am learning to program PHP";
  echo $myText;
  $myText = substr($myText, 5, 8);
  echo "<br />";
  echo $myText;
?>
```

Output:

```
I am learning to program PHP
learning
```

strtolower

This function returns a given string in lowercase:

```
<?php
  $myText = "I am learning to program PHP";
  echo $myText;
  $myText = strtolower($myText);

  echo "<br />";
  echo $myText;
?>
```

Output:

```
I am learning to program PHP
i am learning to program php
```

strtoupper

This function returns a given string in uppercase:

```
<?php
  $myText = "I am learning to program PHP";
  echo $myText;
  $myText = strtoupper($myText);
  echo "<br />";
  echo $myText;
?>
```

Output:

```
I am learning to program PHP
I AM LEARNING TO PROGRAM PHP
```

strpos

With this function, you can search for a particular character or string within a text. This is used, for example, in combination with the above stated function substr practical. The first parameter you need to pass is the text to be searched, while the second one is the string to be searched.

It is entered in the format strpos(string, search term [, from character #]).

Here's an example:

```
<?php
  $myText = "I am learning to program PHP";
  echo strpos($myText, "learn");
?>
```

Output:

```
5
```

strrpos

It is entered in the format strrpos(string, search term, [, from character #]).

This function is used as strpos to find a specific string within a text, but the last occurrence of the string to be searched is returned:

```
<?php
  $myText = "I am learning to program PHP";
  echo strrpos($myText, "m");
?>
```

Output:

```
23
```

strlen

This function returns the length of a string.

```
<?php
  $myText = "I am learning to program PHP";
  echo strlen($myText);
?>
```

Output:

```
28
```

explode

Sometimes, you would use the explode() function in your projects. This splits a string by a delimiter into an array. You could, for example, multiply values from a text file with the piped | isolated standing in a row, and then read as follows disassemble:

```
<?php
  $myText = "value1|value2|value3";
  $valueeArray = explode("|", $myText);
  var_dump($valueeArray);
?>
```

Output:

```
array(3) { [0]=> string(5) "value1" [1]=> string(5) "value2" [2]=> string(5)
"value3" }
```

implode

With the implode function, you can accomplish exactly the opposite: You can create a string from an array, while each value has the possibility to separate, on request, by means of a separator (glue).

```
<?php
  $meinArray = array("Hello", "World,", "how", "are", "you?");
```

```
  $text = implode(" ", $meinArray);
  echo $text;
?>
```

Output:

```
Hello World, how are you?
```

Write your own functions

We have already learnt a lot of useful functions that are included in PHP. But it is also possible to write your own functions. A PHP function must always consist of a name and optionally specified parameters.

Here is an example of two functions to which we give a numerical value and a value for the sales tax. So a net to gross amount and vice versa is calculated:

```
<?php
  function toNetto($amount, $vat_percent = 20) {
    $amount_netto = $amount / (100 + $vat_percent) * 100;
    return $amount_netto;
  }

  function toBrutto($amount, $vat_percent = 20) {
    $amount_brutto = $amount * (100 + $vat_percent) / 100;
    return $amount_brutto;
  }

  $amount = 1000;
  $amount_brutto = toBrutto($amount);
  echo $amount_brutto;
  echo "<br />";
  echo toNetto($amount_brutto);
?>
```

Output:

```
1200
1000
```

In this example, we have included some of the same special features of functions. A function is defined with the preceding word function. Our two functions have two parameters each: One is for the amount and the tax rate; the second parameter is present in the function definition, which is already filled with the value 20. This means that the parameter is optional and if it is not specified when calling the function, the value specified in the definition is used.

Let us, for example, convert an amount of the Austrian VAT (20%) to the German VAT (19%), which goes with our two functions easily:

```
<?php
  function toNetto($amount, $vat_percent = 20) {
  $amount_netto = $amount / (100 + $vat_percent) * 100;
  return $amount_netto;
 }

 function toBrutto($amount, $vat_percent = 20) {
  $amount_brutto = $amount * (100 + $vat_percent) / 100;
   return $amount_brutto;
 }
```

```
  $price_austria = 1256;
  $price_netto = toNetto($price_austria);
  $price_germany = toBrutto($price_netto, 19);
  echo round($price_germany, 2);
?>
```

Output:

```
1245.53
```

With the round() function at the end we make the number better readable.

Split the code in several files

So far, we have written only short scripts of less than 100 lines. Once you work on larger projects that are partly made up of tens of thousands of lines of code, it can soon become confusing within a file. It makes more sense, for example, to outsource function collections in another .php file and integrate them into our main project file.

So, we create two files:

functions_accounting.php

```
<?php
  function toNetto($amount, $vat_percent = 20) {
    $amount_netto = $amount / (100 + $vat_percent) * 100;
    return $amount_netto;
  }

  function toBrutto($amount, $vat_percent = 20) {
    $amount_brutto = $amount * (100 + $vat_percent) / 100;
    return $amount_brutto;
  }
?>
```

vat_calculator.php

```
<?php
  include "functions_accounting.php";

  $price_austria = 1256;
  $price_netto = toNetto($price_austria);
  $price_germany = toBrutto($price_netto, 19);
  echo $price_germany;
?>
```

If, for example, in addition to our vat calculator.php, the file functions of accounting.php are located in the folder ‘functions’, we would have to specify it with the following:

```
<?php
  include "functions/functions_accounting.php";

  $price_austria = 1256;
  $price_netto = toNetto($price_austria);
  $price_germany = toBrutto($price_netto, 19);

  echo $price_germany;
?>
```

You could also specify full absolute paths on the server, such as the following:

```
<?php
  include "/var/www/web1/html/functions/functions_accounting.php";
?>
```

However, if you do this, the path may never change and you may have problems when you

transfer your script to a different server.

Day 4

Storing and reading information with a database (MySQL)

Now, we can query, process and output a lot of information. However, we still lack an important feature of programming: the storage of data and a technique to retrieve it in future. Most PHP projects use an SQL database.

Within each database, data is stored in tables. Like MS Excel, each table consists of individual columns and rows. We will learn how to access the individual values in this chapter.

Connect to a database within PHP

First, we need to connect to a database server from our PHP script.

If you have not modified the basic installation of the server, as described in the first chapter, then the database server access is possible using the user root and without a password. Otherwise, you may need to assign a password in the server management software.

Here is an example of how to connect and reconnect to the database server:

```
<?php
  $mysqli=new mysqli('localhost','my_user','my_password','my_db');
  $mysqli->close();
?>
```

The object, MySQL, expects as parameter, the server, the username on the MySQL server, and the password. The server is usually localhost if the MySQL server is installed on the same machine as the Apache web server.

By default, MySQL connections are automatically closed at the end of a script, so $mysqli->close() is not necessary. If we want to program cleanly and know that from a certain point in the script no database connection is necessary, we should close it, as soon as possible, to save resources.

Create a new database

If we are successfully connected to our database server, we need to select a database to work with. Most web developers usually work with a graphical database management tool. With XAMPP, MAMP and so on and the most preconfigured webserver offered on the Internet, PhpMyAdmin is installed by default. With this, you can easily create and manage new databases, tables and the like. Default t is available at http://localhost/phpmyadmin/or http://localhost/phpMyAdmin/.

However, we would like to try to create our new database directly within our PHP script:

```
<?php
  $mysqli=new mysqli('localhost','root','');
  $result= $mysqli->query("CREATE DATABASE `test`;");
  $mysqli->close();

  var_dump($result);
?>
```

You already know the functions of connecting and disconnecting to the MySQL server. The new function in the example is $mysqli->query. By queries, you tell MySQL what to do. Such queries, as in our example, can also be commands or instructions to update something in our database. However, a result is always returned as to whether the request could be successfully executed.

If you run the script more than once, you will receive a success message only on the first run, then there will be error messages indicating that the given database name has already been assigned.

Select our database

If we know what our database is called, we need to specify the script to use it:

```
<?php
  $mysqli=new mysqli('localhost','root','', 'test');
?>
```

The database, which should be used, is defined as the last parameter. All following queries are now sent to the selected database.

Create tables

Now we want to create a table, for example, to store personal records:

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query("
 CREATE TABLE IF NOT EXISTS `persons` (
`id` int(10) unsigned AUTO_INCREMENT,
`firstname` varchar(50),
`lastname` varchar(50),
PRIMARY KEY (`id`)
 )"
 );

  var_dump($result);
?>
```

This query is a bit more complex. We tell MySQL to create a table named persons, if it does not already exist. When creating the columns, we must already specify which type they will be. Unlike PHP, the database server must know exactly whether a value is a text, a number, a date or any other given type.

We create the column ID of type Integer(10). Integers are numbers without commas. The unsigned attribute means that no sign is to be stored, so there will be no minus sign. Furthermore, we use AUTO_INCREMENT to indicate that the ID column automatically increases by the value of 1 compared to the last highest entry when we insert a new row into the table. Thus, we have a consecutive number that we can use as the primary key as we

specify it in the last line of the table. Each table must have a primary key. This must be unique for each table row so that the primary key allows you to always address a specific table row. The number ten in the brackets (10) defines that the number can have the length of maximum ten digits, so, the maximum number is 9,999,999,999.

We also create a column for the first names and one before the last names of both the variable character (VARCHAR) types. This means that we do not yet know what the length of the string, to be stored, will be, but we limit it to a maximum of 50 characters. Alternatively, we could use the type, TEXT, which has no limitation on a certain number of characters but consumes considerably more memory.

Write data to the database

After we have created our table, we will now fill it with the first data:

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query("INSERT INTO persons (firstname, lastname) VALUES
('Hans', 'Meier');");
  var_dump($result);
?>
```

If you, now, look at the table in PhpMyAdmin, you will see the entry you just added.

To insert data into a table, there is the INSERT INTO SQL statement, followed by the table name and the fields to be filled, which is followed by the VALUES statement, and the values in the same order as the fields specified. If you would like to insert more than one line into the table, you only need to write 'VALUES' once and separate the individual line values with the words:

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query("INSERT INTO persons (firstname, lastname) VALUES
('Max', 'Meier'),('Maria', 'Mauser');");
  var_dump($result);
?>
```

Retrieve data from the database

After we have written some lines in the table, we would like to query them again in our script. This is done with the SQL statement, SELECT:

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query("SELECT firstname, lastname FROM persons;");

  while ($row = $result->fetch_array()) {
    echo $row["firstname"]. " " . $row["lastname"] . "<br />";
  }
```

```
?>
```

The SELECT statement will still be your faithful companion when you write PHP / SQL scripts. A data query using SELECT is followed by the table columns; alternatively you could also write a *, if you want to query all table columns. This is followed by the FROM statement and the name of the table from which the data is to be queried.

After we have submitted the query, we use the function, $result-> fetch_array() to run the result until we can no longer get any more rows. From each line, we access the entries, first name, and last name.

At this point, it is important to know how to access a particular entry in the table by optionally extending the SELECT statement with a condition clause, WHERE:

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query('SELECT firstname, lastname FROM persons WHERE
firstname = "Hans";');

  while ($row = $result->fetch_array()) {
    echo $row["firstname "]. " " . $row["lastname"]. "<br />";
  }
?>
```

Now, we get only rows returned, where in the column, first name, the value is Hans. Likewise, we could now query a specific ID. With this knowledge, you already have the ability to permanently store data and process it with another PHP script in the future.

There are hundreds of books to understand the many other SQL statements and how to create efficient databases or resource-conserving queries. This is very important especially for projects with hundreds of thousands of table rows and dozens of queries per minute. At the end of the book, you will find some good tips and links to free knowledge sources about SQL.

Delete data from the database

The deletion of data f a database is nearly the same SQL statement like the SELECT. You can choose which data should be deleted by filtering it with the WHERE clause.

```
<?php
  $mysqli=new mysqli('localhost','root', '','test');

  $result = $mysqli->query('DELETE FROM persons WHERE firstname = "Hans";');
?>
```

All lines from the database, where the field firstname is Hans, will be deleted. So be careful to double check your WHERE clause.

File operations (read / write files)

In order to import text files into PHP or write to them, they have to be integrated into the script with the use of a filehandler:

```
<?php
  $myFile = "myfile.txt";
  $filehandler = fopen($myFile, "r");
?>
```

The fopen function returns the desired filehandler. As a first parameter, the function expects a filename if the file is in the same directory as the PHP script or the relative path to the file. As a second parameter, the function expects the access mode to the file. This specifies whether we want to have read access with a filehandler for a file or whether we can describe or alter it with our own content.

These are the following access modes for fopen():

'r' Read only, place the file pointer at the beginning

'r+' Read and write, place the file pointer at the beginning

'w' Write only, place the file pointer at the beginning; if the file exists, then empty it

'w+' Read and write, place the file pointer at the beginning; if the file exists, then empty it

'a' Write only, place the file pointer at the end

'a+' Read and write, place the file pointer at the end

'x' Write only, if the file already exists, then return an error

'x+' Read and write, if the file already exists, then return an error

'c' Write only, place the file pointer at the beginning; if the file exists, then do not empty

'c+' Read and write, place the file pointer at the beginning; if the file exists, then do not empty.

Save to a text file

In the following script, the PHP function, frwite(), is used, which first gets our filehandler, and as a second parameter, we have to specify the content which should be added to the file.

To create a new file or extend an existing one, we could use the following script:

```
<?php
  $myFile = "myfile.txt";
  $filehandler = fopen($myFile, "a");

  fwrite($filehandler, "I am a new line.\nI am a second new line.\n");

  fclose($filehandler);
?>
```

In the directory where your .php script file is saved, a new file is created. If you look at the text file now, you will notice that it has been filled by two lines. Everytime you call the php script, two new lines are added. Line breaks must be passed to fwrite using \n.

Reading from a text file

We would like to read this file now into our php script:

```
<?php
  $myFile = "myfile.txt";
  $filehandler = fopen($myFile, "r");
  $contents = fread($filehandler, filesize($myFile));
  fclose($filehandler);

  echo nl2br($contents);
?>
```

In the above script, the PHP function, fread(), is used, which first gets our filehandler, and as a second parameter, we have to specify how much of the file should be read.

We simply pass the size of the file, to be read, using the practical filesize() function. With the function, nl2br(), the line endings from a text file can be converted to HTML-
.

Manage sessions

If we want to recognize a returning visitor, it is necessary to use the so-called sessions. Within sessions, variables can be stored, and they can be used across files. If we start a session for a visitor, a unique identification number, the session ID, is generated.

If we want to open a session for a visitor, it is enough to simply call the function session_start(). You can then set values in the $_SESSION global array. If the same session is to be reused on the next call, it is sufficient to call it session_start() at the beginning of the script. In the visitor's browser is stored a cookie with the session ID; the values of the $_SESSION array are stored on the server in a temporary cache. If you want to end the session securely, you must use the session_destroy() function.

Here is an example of how to use sessions:

session1.php

```
<?php
  session_start();
  $_SESSION["username"] = "Walter";
  echo "Hello " . $_SESSION["username"];
?>
```

session2.php

```
<?php
  session_start();
  if (isset ($_SESSION["username"])) {
    echo "Hello " . $_SESSION["username"];
  }
  else {
    "First please open the page session1.php!";
  }
?>
```

After the call of session_start() in the second file, the same content of the $_SESSION array is usable as in the first file.

Day 5

Write clean code (which is still understandable in a year)

Before you acquire an unclean programming style, I would like to give you some tips on how to write a ‘clean’ code on your way as a programmer.

On the one hand, it is extremely important, as has already been described in the chapter on variables, to use appropriate and self-describing variables and function names. Nothing is more disappointing than to read variables like $a, $b, or $x in an unknown code because they do not show what will be stored in them.

Use tab stops in your code writing. Individual programming studios sometimes have a somewhat different form, but many OpenSource projects have the following syntax style:

```
function name (parameter) {
  code block
}
```

That is, a space between function name, a space after the parenthesis with the parameters followed by a curved parenthesis. The code block of the function is inserted by a tab step, and the closing curved bracket is again at the height of the function name. This makes it very easy to see where a function starts and stops.

Do not save with comments. You do not have to comment on any one-line If query, but a certain structure for longer scripts is essential.

Do not write a code twice, that is, avoid redundancy. Once a code is used several times, write a function. This increases the clarity of the code and saves valuable memory space. In addition, you need to make corrections or extensions only at one code point.

Security within PHP scripts

Here are some of the most important tips to give you some idea on how to make your PHP scripts safer:

Principle: Never trust data coming from outside

If you follow this principle, you have already won half the battle against crackers and hackers. Regarding the mistrust of external data, I mean you should validate the input data to see if it corresponds to the expected format. This means that, if a number is to be entered, it is best to check if the function is numeric ($input), whether it is really a number, or whether an e-mail address is set up as it should be (exactly one @ character and a domain must be specified at the end). This brings us to the next topic.

Protection against SQL Injections

An SQL injection is a way manipulate an SQL Query so that it does something other than what it was meant for. This happens if you do not filter the entered data.

Suppose we have the following script to offer a login functionality for a web page:

```
<?php
  $username = $_GET["username"];
  $password = $_GET["password"];

  $sql = "SELECT * FROM users WHERE username = "".$username."" AND password = ""
. $password ."" ";

  $mysqli=new mysqli('localhost','root','', 'test');


  $result = $mysqli->query("$sql);

  echo "Found the following user(s): ";
  while ($row = $result->fetch_array()) {
    echo $row["username"]. "<br />";
  }
?>
```

This functionality has no problem if the users are just entering their username or password. But, unfortunately, there are technically savvy crackers who look for such simple logins, some use automated bots to find these logins, and try to enter the following:

```
Username: admin
Password: " OR 1=1
```

If we pass this text unfiltered into our SQL query, we get the following SQL statement:

```
SELECT * FROM users WHERE username = "admin" AND password = "" OR 1=1
```

Since 1 is always equal to 1, a result is always returned. So the cracker is logged in as admin or even adds his users:

```
Username: x
Password: x"; INSERT INTO users (username, password) VALUES ("mynewuser",
"mypassword");
```

The SQL would be:

```
SELECT * FROM users WHERE username = "x" AND password = "x"; INSERT INTO users
(username, password) VALUES ("mynewuser", "mypassword");
```

Both SQL statements would be executed and the cracker would have already created a new user account. Theoretically, he could also send a DELETE command and thus destroy our database or, for example, change the password of the administrator.

But how can you protect yourself against this?

Either you filter the specified data for only allowed characters (a-z, A-Z, 0-9) or you 'escape' the input data. This means, all special characters as well as the entire string are encapsulated by apostrophes and are thus made 'harmless'. There is the ready-made function $mysqli->real_escape_string(), which gets a string as parameter and returns a safe escaped version of it.

Example:

```
$password =$mysqli->real_escape_string($password);
```

Keep your passwords encrypted

Something even large global companies have had to learn through bitter experience the need to always store passwords always encrypted in the database. There are encryption methods that encrypt only in one direction and generate so-called hashes from arbitrarily long strings. These are strings of a previously known length. PHP delivers its own well tested hashing library since PHP 5.5. You can use it e.g. with the function password_hash($password, ALGORITHM).

```
<?php
  $password = "password123";
  $hash = password_hash($password, PASSWORD_BCRYPT);
  echo $password;
  echo "<br />";
  echo $hash;
?>
```

Output (the hash may differ from server to server and generated every time):

```
password123
$2y$10$K6JP7Lfu2.wjuEGLzCact.8GZaok5oSWhH49.TzTZm5/PPfDzXYMW
```

The hash can no longer be deciphered in plaintext. Now you can save these password hashes in your database. Every time a user wants to log in, you use the password_hash() function to generate the hash for the data the user put in. If the hashes are the same, the user entered the correct password. If a hacker wants to steal your database, he would need a lot of

time to try all his word lists, because the password_hash() function needs some time to generate the hashes. You can add an optional $options parameter and define that a more complex method of hashing should be used so that the hashing gets even slower and is harder to crack.

Add salt to your passwords

Another security aspect is to protect oneself against users with unsafe passwords. Statistics from large user databases show that a considerable percentage of the users always use the same passwords, as, for example, asdf123, password, 1234567890. If a cracker gets our username with the hashes, it would be easy for him to match the hashes against the 1000 most commonly used passwords on the Internet. To prevent this, I recommend adding a long random string for each project, such as ‘afdsoin23oidvnio3iosdvn09832jknlsd’.

If a user logs in again and uses the password ‘password’, we store the hash for ‘passwordafdsoin23oidvnio3iosdvn09832jknlsd’. With each login, we attach this string to the entered password in our script and calculate the hash. If someone is collecting our user database, it is impossible for him to check the hashes if he does not know our string.

The password_hash() function already adds a random salt automatically.

Website tips

PHP

www.php.net/manual/en/

Official guide to PHP with a good documentation on all functions existing in PHP, including numerous examples, is always updated when a new PHP version is released.

www.w3schools.com/php/

A very good tutorial site, explaining all available functions and

SQL

dev.mysql.com/doc/

Official site with large functional reference, but with somewhat complex examples

www.php.net/manual/en/book.mysqli.php

Part of the official PHP guide/documentation with all PHP functions available around MySQL

forums.mysql.com

Official MySQL forum with a very active community

Book tips

All of the following books are also available as e-books.

PHP and MySQL for Dynamic Web Sites, Fourth Edition: Visual QuickPro Guide
by Larry Ullman
Peachpit Press
ISBN 0321784073
A good book covering most of the PHP and MySQL functions. Try to get the newest version, as there are a lot of changes in the latest PHP releases.

Head First PHP & MySQL: A Brain-Friendly Guide Kindle Edition
by Lynn Beighley and Michael Morrison
O'Reilly Media
ISBN-10: 0596006306
ISBN-13: 978-0596006303
An easy to understand book on PHP and MySQL. I hope there is a new version released soon to cover the PHP 7 extras.

PHP Objects, Patterns, and Practice
by Matt Zandstra
ISBN-13: 978-1430260318
ISBN-10: 1430260319
A good book on programming patterns with focus on PHP. You will learn how to write clean code with standardized programming patterns. A very important topic in programming is the development of object-oriented projects. In any case, it is recommended as soon as you are familiar with the basics of PHP.

Wikibook PHP Programming
https://en.wikibooks.org/wiki/PHP_Programming
Wikibooks is a Wikipedia-based platform where, similar to the Wikipedia encyclopaedia, a large community writes non-fiction texts that are edited in book form. Some of the books have even appeared in print versions. The online Wikibooks have the advantage, thanks to the active community, that they are almost ‘live’ and kept up-to-date.

Epilogue

I hope you enjoyed the introduction to the world of web development with PHP and MySQL and you are now ready to implement your first own small projects.

If you need ideas about to try next: you could use the knowledge this course provides, for example, to create a guest book for a website, or a small script that displays a different greeting, depending on the day of the week and the time.

Or, how about having a page for collecting signatures or votes? The page could write the data to a database, but it should check beforehand whether an e-mail address and/or a certain combination of first and last names already exists.

Do you have any feedback on this book? I would be very pleased.

I am available at the e-mail address books@theklaus.at.