



PHP-Einführung

Jürgen Lange

„Die Technik, welche weder gut noch böse ist, ist ohne Bezug zur Moral. Die Moral steckt nicht in dem Hammer, sondern in dem Menschen, der ihn führt. Die Technik bedarf einer moralischen Instanz, welche eine Kontrolle über ihre Anwendung zum Nutzen des Menschen ausübt.“

Peter Bamm

In diesem Kapitel erwerben Sie grundsätzliches PHP-Wissen. Sie werden in Vorgehensweisen von der Installation bis zum Schreiben eigener Funktionen eingeführt und erhalten anhand von Beispielen einen kurzen Einblick in die praktische PHP-Programmierung.

2.1 Was ist PHP?

PHP ist im Grunde genommen nichts weiter als eine Skriptsprache zur Erstellung von dynamischen Websites. Es wurde 1994 von Rasmus Lerdorf in den ersten Versionen entwickelt und hat sich seitdem zu einer der sich am meisten und schnellsten verbreitenden Programmiersprachen im Web gemausert.

Die Funktionsweise von PHP unterscheidet sich im groben Vergleich zu HTML nur darin, dass PHP nicht wie eine HTML-Seite einfach vom Server heruntergeladen wird, sondern vor dem Download an den auf dem Server installierten PHP-Parser gesendet wird. Der Parser verarbeitet den PHP-Code dann entsprechend und gibt die gewünschten Werte zurück.

2.2 Installation mit Hilfe von PHPTriad

Die Installation von PHP stellt für viele Einsteiger eine große Hürde dar. Es gibt bereits viele Tutorials zur Installation und Konfiguration des Apache-Webserver inklusive PHP und MySQL.

PHP arbeitet natürlich nicht ausschließlich mit dem Apache-Webserver, wird aber am häufigsten in dieser Kombination genutzt (WAMP – Windows, Apache, MySQL, PHP). Auch aus diesem Grund haben wir uns entschlossen, auf diese Installation noch einmal genauer einzugehen. Über andere Installations-Tutorials informiert Sie die Info-Box.



Weitere Installationsmöglichkeiten finden Sie gesammelt unter <http://www.dynamic-webpages.de/php/installation.php>!

Einige Beispiele:

WAMP: <http://www.dynamic-webpages.de/php/install-windows95-nt.php> – Installation auf Win95/Wing8/NT

UNIX: <http://www.dynamic-webpages.de/php/install-unix.php> – Installation auf UNIX-Systemen

Was ist PHPTriad?

PHPTriad ist eine ausführbare Installationsroutine, die sowohl Apache, PHP, MySQL (einschließlich des Verwaltungstools PHPmyAdmin) und sogar Perl einfach und funktionsfähig installiert. Beachten Sie, dass PHP nicht im CGI-Modus installiert wird, was man aber zur Not noch ändern könnte. Mehr dazu erfahren Sie unter den oben angegebenen Ressourcen.

Die Installation

Zunächst laden Sie bitte die aktuellste Version von PHPTriad unter <http://sourceforge.net/projects/phptriad/> herunter. (Dort finden Sie die aktuellste Version, auch wenn sich die Projektseite an sich unter <http://www.phpgeek.com> befindet.)

Der Nachteil solcher Installationspakete ist, dass zumeist nicht die aktuellsten Versionen der einzelnen Distributionen darin enthalten sind. Demzufolge sehen Sie sich die einzelnen Versionen der enthaltenen Module genau an und updaten Sie gegebenenfalls einzelne Teile. Dazu folgende Hinweise:

PHPmyAdmin

Aktuellste Version unter <http://phpmyadmin.sourceforge.net/>

Nach Installation von PHPTriad: Einfach den Ordner entpacken und im Document-Root (siehe ► HTTPD.CONF) überschreiben bzw. in das Document-Root hinein kopieren.

PHP-Update

Aktuellste Version unter http://sourceforge.net/project/showfiles.php?group_id=9325&release_id=69192
Nach Installation von PHPTriad: Downloaden und .exe-Datei ausführen! Damit sollten Sie vorerst auskommen. Führen Sie nun die PHPTriad.exe-Datei aus und folgen Sie den Anweisungen am Bildschirm. Die Installationsroutine erstellt einen Ordner APACHE mit folgender Struktur (siehe Abbildung 2.1).



■■■ **Abbildung 2.1:**
Ordnerstruktur nach
der Installation von
PHPTriad (Version 2.1.1)

Konfiguration von PHP und MySQL

Die für uns wichtigen Ordner sind /CONF, /HTDOCS, /PHP und /MYSQL.

► HTTPD.CONF

Beginnen wir mit dem Ordner /CONF. Darin befindet sich eine Datei namens HTTPD.CONF, die Sie nun bitte öffnen. Sie sehen sehr viel Text, der hauptsächlich Erläuterungen beinhaltet. Achten Sie nunmehr nur auf die Zeilen ohne #-Zeichen davor.

```

httpd.conf - Editor
Datei Bearbeiten Format Ansicht ?
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents.
ServerAdmin lange@netzproduktion.com

#
# ServerName allows you to set a host name which is sent back to clients for
# your server if it's different than the one the program would get (i.e., use
# "www" instead of the host's real name).
# Note: You cannot just invent host names and hope they work.  The name you
# define here must be a valid DNS name for your host.  If you don't understand
# this, ask your network administrator.
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address (e.g., http://123.45.67.89/)
# anyway, and this will make redirections work in a sensible way.
# 127.0.0.1 is the TCP/IP local loop-back address, often named localhost.  Your
# machine always knows itself by this address.  If you use Apache strictly for
# local testing and development, you may use 127.0.0.1 as the server name.
ServerName localhost

#
# DocumentRoot: The directory out of which you will serve your
# documents.  By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
DocumentRoot "d:/Projekte/"

#
# Each directory to which Apache has access, can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
# First, we configure the "default" to be a very restrictive set of
# permissions.
<directory />
    options FollowSymLinks
    AllowOverride None
</directory>

```

■ ■ ■ **Abbildung 2.2:** Die Apache-Konfigurationsdatei httpd.conf

Tragen Sie bei SERVER-ADMIN Ihre E-Mail-Adresse ein. Sie wird beispielsweise später beim Aufrufen einer fehlerhaften Seite bzw. einer nicht vorhandenen Seite angezeigt.

ServerAdmin lange@netzproduktion.com

Unter DOCUMENT ROOT tragen Sie den Ordner ein, welchen Sie als Projekt-Ordner für PHP-Dateien verwenden. Dies kann zum Beispiel

```
DocumentRoot "D:/_Projekte/"
```

oder auch

```
DocumentRoot "C:/"
```

sein. Der Nachteil der zweiten Variante ist, dass Sie später beim Aufruf des LOCALHOST immer den Pfad zu Ihrer entsprechenden PHP-Datei angeben müssen. Dies ist zeitaufwändiger, als wenn Sie den tiefstmöglichen Ordner (eben den Projekt-Ordner) auswählen würden. Außerdem ist es nicht sinnvoll, als Document-Root den Festplatten-Root anzugeben, da dadurch die Sicherheit bei gleichzeitiger Online-Aktivität leidet!

Der nächste Schritt besteht darin, auch die <DIRECTORY>, etwa drei Absätze weiter unten, auf das Document-Root zu aktualisieren. Tragen Sie dort also dasselbe ein wie unter DOCUMENT ROOT.

```
<Directory "D:/_Projekte/">
```

Für die Standardkonfiguration sollten diese drei Änderungen vorerst ausreichen. Speichern und schließen Sie also die Datei und öffnen Sie nun im Ordner /PHP die Datei PHP.INI.

► PHP.ini

Dies ist die Konfigurationsdatei für den PHP-Parser. Scrollen Sie langsam bis zum Absatz [SESSION].



■ ■ ■ **Abbildung 2.3:** *php.ini – Konfigurationsdatei für den PHP-Parser*

Dort verändern Sie den `SESSION.COOKIE_PATH` auf `C:/APACHE/PHP/SESSIONS`.

Als Nächstes legen Sie im Ordner `/PHP` eben diesen Ordner `/SESSIONS` an. Hierin werden die PHP-Session-Dateien abgelegt. Dazu aber später mehr.

Haben Sie das soweit erledigt, speichern und schließen Sie wiederum die Datei `PHP.INI`.

► MySQL

Abschließend konfigurieren Sie nun das MySQL-Datenbankmodul.

Sie finden im Ordner `/MYSQL/BIN` eine Datei mit dem Namen `WINMYSQLADMIN.EXE`, die Sie bitte öffnen.



■ Abbildung 2.4: Die Datei `winMySQLadmin.exe`

Sie werden nun nach einem Benutzernamen und einem Passwort gefragt, die Sie ab sofort als MySQL-Zugangskombination für Ihre Datenbankskripte nutzen werden.



■ Abbildung 2.5: Der `WinMySQLAdmin` mit zugehörigem Passwort und Benutzernamen

Als Nächstes sollten Sie, sofern noch nicht aktiv, den Server-Standalone und den MySQL-Service starten (siehe Abbildung). Klicken Sie dazu mit der rechten Maustaste und wählen Sie, wie in der Abbildung gezeigt, die Optionen **START THE SERVER-STANDALONE** und **START THE SERVICE**. Dies brauchen Sie nur einmal zu tun, da sich ab sofort MySQL schon beim Start Ihres Betriebssystems aktiviert.



■ ■ ■ **Abbildung 2.6: MySQL-Service und Standalone starten**

Abschließend verstecken Sie den WinMySQLAdmin, indem Sie die Option **HIDE ME** mit Hilfe der rechten Maustaste auswählen. Sie sehen nun in der Taskleiste rechts unten eine Ampel, welche grün leuchten sollte. Sollte dies nicht der Fall sein, klicken Sie im Startmenü unter **PROGRAMMS** auf **PHPTRIAD > MYSQL > MYSQL-D**.

Kopieren Sie den Ordner **/PHPMYADMIN** aus **/APACHE/HTDOCS** in Ihr **DOCUMENT ROOT**.



■ ■ ■ **Abbildung 2.7: Ordnerstruktur im Startmenü (WinXp – PHPTriad 2.2.1) mit verschobenen „PHPmyAdmin“ und „Launch Site“**

► Der Test

Die Installationsroutine hat in Ihrem Startmenü einen Eintrag namens **PROGRAMMS > PHPTRIAD** erstellt. Des Weiteren gibt es die Option **START APACHE** im Unterordner **APACHE CONSOLE**. Klicken Sie diese bitte an.

Es sollte sich ein neues DOS-artiges Fenster öffnen, mit dem folgendem Inhalt: **„APACHE/1.3.23 <WIN23> RUNNING...“** (Windows-Systeme).

Sollte das geschehen sein, öffnen Sie als Nächstes Ihren Standard-Browser (das ist in den meisten Fällen der Internet Explorer). Geben Sie folgenden Link ein: <http://localhost/PHPmyAdmin/>. Es sollte sich Ihnen nun ein Bild ähnlich wie in der Abbildung bieten.



Abbildung 2.8: PHPMyAdmin auf localhost

Dies ist das Zeichen, dass sowohl der Apache als auch MySQL richtig konfiguriert sind.

Öffnen Sie Ihre Skripte von nun an immer nach folgendem Schema (Apache ist gestartet!):

Im Browser öffnen: `http://localhost/+ Pfad zum Skript.`

Beispiel: `http://localhost/Ordner/Unterordner/index.php`

2.3 PHP in diesem Buch

Wir nutzen momentan PHPTriad mit PHP 4.1.1. Darauf bauen wir unsere Skripte im Buch auf. Der größte Teil funktioniert aber auch mit PHP-Releases unter PHP 4.0.

Es empfiehlt sich natürlich immer, die aktuellste PHP-Version zu nutzen. Die momentan aktuellste PHP-Version ist 4.1.2. Hierbei handelt es sich um einen Bugfix-Release, in dem unter anderem eine Sicherheitslücke in Multipart/Form-Data POST Requests geschlossen wurde.

2.4 Guter Code

Bevor wir mit dem Programmieren erster Skripten beginnen, möchten wir auf die allgemein anerkannten Konventionen für guten Code eingehen. Denn sollte es einmal vorkommen, dass Sie im Team arbeiten oder andere Leute Ihren Code lesen müssen, so erleichtert guter Code die Arbeit enorm. Kommentare, Einrückungen und eindeutige Variablenamen sind nur einige Schlagwörter in diesem Zusammenhang.

Kommentare

Kommentare sind die wichtigsten Anhaltspunkte für andere Programmierer, um Ihren Code zu verstehen. Geizen Sie nie mit Kommentaren.

Kommentare setzen Sie innerhalb der PHP-Skripte, möglichst immer mit einer Zeile Abstand, auf verschiedene Weisen. PHP kennt drei Varianten:

```
<?php
// Dies ist ein einzeiliger, am häufigsten verwendeter Kommentar

/* Dies ist ein mehrzeiliger Kommentar, welcher erst wieder durch
das entsprechende Zeichen am Ende des Kommentars beendet wird */

# Dies ist wiederum ein einzeiliger Kommentar

?>
```

■ ■ ■ *Listing 2.1:*
Kommentare für guten
Code verwenden

Wenn Sie Kommentare setzen wollen, sollten Sie sich immer fragen:

- Was tue ich gerade?
- Wo tue ich es?
- Warum auf diese Weise?
- Warum an dieser Stelle?
- Wie wirkt sich das auf andere Programmteile aus?

Einrückungen

Einrückungen sind im Prinzip nichts anderes als die Verschachtelung zusammengehöriger Skriptmodule, die das Lesen des Codes aber wesentlich erleichtern. Das folgende Listing müssen Sie noch nicht verstehen, es soll lediglich die Verschachtelung demonstrieren.

Listing 2.2: ■■■
Demonstration des Einrückungseffekts. Innerhalb der geschweiften Klammer wird alles nach rechts versetzt eingerückt.

```
<?php
$tag = date("w");
switch ($tag) {
    case 0: echo "Sonntag"; break;
    case 1: echo "Montag"; break;
    case 7: echo "Sonntag"; break;
    case 6: echo "Samstag" ; break;
    case 5: echo "Freitag"; break;
    case 4: echo "Donnerstag" ; break;
    case 3: echo "Mittwoch"; break;
    case 2: echo "Dienstag"; break;
    default: echo "komischer Tag";
}

echo date(", d.m.Y ");

?>
```

Wie Sie sehen, ist der Effekt deutlich zu erkennen. Obwohl Sie den Code vielleicht noch nicht verstehen, wissen Sie trotzdem, dass alles innerhalb der geschweiften Klammern zur Funktion SWITCH() gehört.

Eindeutige Variablennamen

Wenn die Skripte länger werden, wächst auch die Anzahl der Variablen. In PHP müssen Sie Ihre Variablen nicht vorher deklarieren, wie es zum Beispiel in Objekt-Pascal (Delphi) nötig ist. Sie brauchen auch keine Datentypen für Ihre Variablen zu vergeben. PHP sucht sich automatisch den am besten geeigneten Datentyp aus und kann diesen auch innerhalb eines Dokuments beliebig oft ändern.

Eine Konvention bezüglich Variablen sollten Sie allerdings immer einhalten. Die Vergabe von Variablennamen gehört zu den häufigsten Aufgaben eines Programmierers, wird aber am wenigsten reflektiert.

Sie sollten Ihre Variablen wie ein Telefonbuch führen. Sie sollten immer wissen, wofür die Variable zuständig ist, sobald Sie sie nur einmal, selbst ohne zugehörigen Code, lesen. Es haben sich natürlich mehrere Schemata für die Vergabe von Variablennamen etabliert. Man kann grob unterscheiden in kurze und sprechende Variablennamen. Beispiele dafür sind:

Listing 2.3: ■■■
Verschiedene Arten von Variablennamen

```
<?php
$name_angestellter_firma_netzproduktion = "Jürgen Lange";
$name_modul1 = "Jürgen Lange";

?>
```

Welches System Sie letztendlich nutzen, kann Ihnen niemand vorschreiben, Sie müssen Ihren eigenen Programmierstil entwickeln und mit ihm zufrieden sein. Es gibt schließlich auch Menschen, die ein komplettes Programm in eine Zeile schreiben und damit besser klarkommen, als wenn sie nach den strengsten Konventionen programmiert hätten.

Quintessenz

Das Wichtigste, wie auch immer Sie programmieren, ist Kontinuität. Kontinuität ist die oberste Direktive, denn auch der undurchsichtigste Code wird irgendwie durchschaubar, sofern er konstant dem gleichen Stil folgt.

Des Weiteren könnten folgende Leitsätze der Programmierung Ihnen den Einstieg erleichtern:

- Machen Sie Ihren Code leicht lesbar.
- Fügen Sie, wo möglich, Kommentare hinzu.
- Wählen Sie sprechende Variablennamen.
- Unterteilen Sie Ihren Code in logische Funktionsgruppen.
- Trennen Sie separate Teile des Codes ab.
- Erstellen Sie eine Dokumentation.

2.5 PHP und HTML

Allgemeine Syntax

PHP-Skripte werden, wie Sie sicherlich schon bemerkt haben, mit bestimmten Zeichenfolgen eingeleitet und beendet. Auch hier gibt es wieder verschiedene Möglichkeiten:

► Variante 1 – XML-Stil (Extensible Markup Language)

```
<?php
// Hier kommt PHP-Code
?>
```

Diese Variante ist die am häufigsten verwendete Groß- und Kleinschreibung hinter dem ? wird nicht beachtet, sprich <?php funktioniert genauso gut wie <?PHP.

► **Variante 2 – SGML-Stil (Standard Generalized Markup Language)**

```
<?
// Hier kommt PHP-Code
?>
```

Hier gilt dasselbe wie für Variante 1, nur ohne PHP-Spezifizierung.

► **Variante 3 – ASP-Stil (Active Server Pages)**

```
<%
// Hier kommt PHP-Code
%>
```

Diese Variante setzt den Wert *ON* bei der Option `ASP_TAGS` in der PHP-Konfigurationsdatei `PHP.INI` – im Absatz *Language Options* – voraus.

► **Variante 4 – JavaScript-Stil**

```
<Skript language="PHP">
//Hier kommt PHP-Code
</Skript>
```

Die meiner Meinung nach umständlichste und deswegen auch am seltensten benutzte Variante.

Einbettung in HTML

Der große Vorteil von PHP ist, dass es sich in jeglichen HTML-Code integrieren lässt. Sie können mittels PHP HTML-Tabellen, Formulare oder Menüs erzeugen. Ein PHP-Skript lässt sich dabei ganz einfach in den HTML-Code einbetten.

Beispiel:

Listing 2.4: ■■■
PHP-Integration in HTML

```
<html>
<head>
<title>Flash und PHP - Integration von PHP in HTML</title>
</head>
<body>
<?php
// Wenn eine gezogene Nummer 0 ist, hat der Kandidat eine Niete
// Wenn die gezogene Nummer 1 oder größer ist, hat der Kandidat
// einen Gewinn
?>
<?php if ($nummer == 0) { ?>
<table width=460 border=0 cellspacing=0 cellpadding=0
align=center>
<tr>
<td>
```

```

        <div align=center> Niete !</div>
    </td>
</tr>
</table>
<?php } ?>
<?php if ($nummer >= 1) { ?>
<table width=460 border=0 cellspacing=0 cellpadding=0
    align=center>
    <tr>
        <td>
            <div align=center> Gewinn !</div>
        </td>
    </tr>
</table>
<?php } ?>
</body>
</html>

```

Auch hier gibt es wieder zwei verschiedene Möglichkeiten, die jeder Programmierer für sich entdecken sollte. Man kann zum einen die PHP-Skripte direkt in den HTML-Code integrieren und vor jeder Ausgabe das Skript wieder beenden, wie im Beispiel oben, oder man erzeugt selbst die Ausgabe per PHP.

Dazu ebenfalls ein Beispiel:

```

<?php

echo"<html><head><title>Flash und PHP - Integration von PHP in
HTML</title></head>";

echo"<body>";

// Wenn eine gezogene Nummer 0 ist, hat der Kandidat eine Niete
// Wenn die gezogene Nummer 1 oder größer ist, hat der Kandidat
// einen Gewinn
if ($nummer == 0) {
echo"<table width=\"460\" border=\"0\" cellspacing=\"0\"
    cellpadding=\"0\" align=\"center\">
    <tr>
        <td>
            <div align=center> Niete !</div>
        </td>
    </tr>
</table>";
}
if ($nummer >= 1) {
echo"
<table width=460 border=0 cellspacing=0 cellpadding=0
    align=center>
    <tr>

```

■ **Listing 2.5:**
Gleiches Ergebnis
durch HTML-Erzeugung
per PHP

```

        <td>
            <div align=center> Gewinn !</div>
        </td>
    </tr>
</table>";
}
echo"</body></html>";
?>

```

Dieses Skript liefert die gleiche Ausgabe wie das erste, sofern die Variable \$nummer identisch ist, es ergibt sich also von der Wirkungsweise her kein Unterschied.

Einige Programmierer arbeiten zur Zeitersparnis bei der Erstellung von HTML mit WYSIWYG-Editoren („What you see is what you get“), wie z.B. Dreamweaver. Dieser unterstützt jedoch nur die erste Variante.

Das Arbeiten mit beispielsweise Dreamweaver hat aber den Vorteil, dass man schnell bestimmte Tabelleneigenschaften ändern kann, was bei Variante 2 ziemlich mühselig werden kann. Wer HTML allerdings sowieso von Hand programmiert, dem wird das egal sein.

Wichtig ist noch zu beachten, dass der PHP-Interpreter nicht mit den Anführungsstrichen innerhalb von HTML zurecht kommt, d.h. Anführungsstriche wie z.B. bei

```

<table width="460" border="0" cellspacing="0" cellpadding="0"
align="center">

```

müssen dann mit einem „\“ versehen werden, damit der Parser Bescheid weiß, dass es sich um ein Sonderzeichen handelt.

```

<table width=\"460\" border=\"0\" cellspacing=\"0\"
cellpadding=\"0\" align=\"center\">

```

PHP-Befehle enden immer mit einem Semikolon ;.

2.6 Variablen

Variablen verweisen auf einen bestimmten Speicherplatz im Arbeitsspeicher des Rechners, der mit beliebigem Inhalt gefüllt werden kann. In PHP werden Variablen auf ganz einfache Weise deklariert, indem man das \$-Zeichen vor eine Zeichenkette setzt. Beim Variablennamen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Des Weiteren entfällt in PHP ebenfalls die explizite Typzuweisung, die beispielsweise bei Objekt-Pascal (Delphi) nötig ist. Variablen entstehen, sobald sie einem Wert zugewiesen werden. Dazu als Beispiel ein einfaches Skript:

```

<?php

$variable1 = 100;
$variable2 = 100.5;

// Addition von $variable1 und variable2
$ergebnis = $variable1 + $variable2;

// Ausgabe des Ergebnisses
echo "$ergebnis";

// Zeilenumbruch erzeugen
echo "<br>";

// Variablen mit Zeichenketten
$autor = "Jürgen Lange<br>";
$buchtitel = "PHP und Flash";

// Verschmelzung von $autor und $buchtitel zu $ausgabe
$ausgabe = $autor.$buchtitel;

// Ausgabe der Variable $ausgabe
echo "$ausgabe";

?>

```

■ ■ ■ Listing 2.6:
Arbeiten mit Variablen

Ausgabe:

```

200.5
Jürgen Lange
PHP und Flash

```

Wie Sie sicherlich am Skript erkannt haben, werden Variablen und damit deren Inhalt nicht sofort ausgegeben. In PHP bedarf es dazu einer Funktion `echo()`, deren Anwendung im unteren Skript nochmals explizit gezeigt wird.

```

<?php

$variable = "Hallo Welt";
echo "$variable<br>";
echo("$variable");

?>

```

■ ■ ■ Listing 2.7:
Hallo Welt

Ausgabe:

```

Hallo Welt
Hallo Welt

```

Die Funktion `echo()` kann sowohl mit Klammern als auch mit einfachen Anführungsstrichen genutzt werden.

2.7 Arrays

Das Thema Arrays ist meiner Meinung nach immer ein kleiner Knackpunkt. Wer das System der Arrays verstanden hat, kann große Probleme damit lösen, wer nicht, der wird sich auf ewig um die Anwendung der Arrays drücken wollen. Aber: Ohne geht's nicht ☹!

Arrays sind Wertelisten, die verschiedene, zusammengehörige Werte enthalten. Ein gutes Beispiel für die Anwendung ist die Speicherung von Mitarbeiternamen einer Firma in einem Array.

Arrays sind gleichermaßen Variablensammlungen, in denen mehrere Variablen gleichen Typs zusammengefasst werden. Sie besitzen Dimensionen, wobei das einfachste Array nur eine Dimension hat, und können beliebig viele Werte aufnehmen.

Indizierte Arrays

Ein Beispiel für indizierte Arrays ist die vorhin angesprochene Speicherung der Mitarbeiternamen einer Firma.

Schauen Sie sich das Listing genau an, es wird Ihnen beim Verstehen des Array-Systems helfen!

Listing 2.8: ■■■
*Verschiedene Formen
des indizierten Arrays*

```
<?php
// Schreibweise der Variablenzuweisung in einem indizierten
Array
$mitarbeiter[] = "Jürgen Lange";
$mitarbeiter[] = "Martin Brochier";
$mitarbeiter[] = "Azad Adsay";
$mitarbeiter[] = "Ulrike Dittmann";

// Gleiches Ergebnis bei dieser Schreibweise
$mitarbeiter[0] = "Jürgen Lange";
$mitarbeiter[1] = "Martin Brochier";
$mitarbeiter[2] = "Azad Adsay";
$mitarbeiter[3] = "Ulrike Dittmann";

// Ausgabe für 1. Beispiel
echo"Ausgabe für <b>1. Beispiel</b><br><br>";

echo"Mitarbeiter 1: $mitarbeiter[0]<br>";
echo"Mitarbeiter 2: $mitarbeiter[1]<br>";
echo"Mitarbeiter 3: $mitarbeiter[2]<br>";
echo"Mitarbeiter 4: $mitarbeiter[3]<br><br>";

// Alternativ mit Schlüsselwort array();
$mitarbeiter = array("Jürgen Lange",
                    "Martin Brochier",
```

```

        "Azad Adsay",
        "Ulrike Dittmann");

echo"Ausgabe für <b>Schlüsselwort array()</b>:<br><br>";

echo"Mitarbeiter 1: $mitarbeiter[0]<br>";
echo"Mitarbeiter 2: $mitarbeiter[1]<br>";
echo"Mitarbeiter 3: $mitarbeiter[2]<br>";
echo"Mitarbeiter 4: $mitarbeiter[3]";

?>

```

Ausgabe:

```

Ausgabe für 1. Beispiel
Mitarbeiter 1: Jürgen Lange
Mitarbeiter 2: Martin Brochier
Mitarbeiter 3: Azad Adsay
Mitarbeiter 4: Ulrike Dittmann

```

Ausgabe für Schlüsselwort array():

```

Mitarbeiter 1: Jürgen Lange
Mitarbeiter 2: Martin Brochier
Mitarbeiter 3: Azad Adsay
Mitarbeiter 4: Ulrike Dittmann

```

Die Funktion `ARRAY()` eröffnet allerdings noch weitere Möglichkeiten, beispielsweise die der Verschachtelung, mit der man Arrays von Arrays (mehrdimensionale Arrays) bilden kann usw., auf die ich etwas später noch eingehen werde.

Assoziative Arrays

Der Unterschied zum indizierten Array ist eigentlich nur der, dass Sie bei assoziativen Arrays mit Schlüsseln, also Wertepaaren (einige von Ihnen kennen das bestimmt noch aus der Schule), arbeiten.

Assoziative Arrays erlauben auch Zeichenketten als Index.

```

<?php

// erstes Beispiel
$mitarbeiter["JL"] = "Jürgen Lange";
$mitarbeiter["AA"] = "Azad Adsay";

// Ausgabe erstes Beispiel
echo "Mitarbeiter mit den Initialen JL ist:
    ".$mitarbeiter["JL"]. "<br>";
echo "Mitarbeiter mit den Initialen AA ist:
    ".$mitarbeiter["AA"];

```

■ ■ ■ *Listing 2.9:*
Erzeugung und
Ausgabe von
assoziativen Arrays

```

// zweites Beispiel
$schueler[Nummer1] = "Jürgen Lange";
$schueler[Nummer2] = "Azad Adsay";

// Umbruch erzeugen
echo "<br>";

// Ausgabe zweites Beispiel
echo "Schüler1 ist ".$schueler["Nummer1"]. "<br>";
echo "Schüler2 ist ".$schueler["Nummer2"];

?>

```

Ausgabe:

```

Mitarbeiter mit den Initialen JL ist: Jürgen Lange
Mitarbeiter mit den Initialen AA ist: Azad Adsay
Schüler1 ist Jürgen Lange
Schüler2 ist Azad Adsay

```

Hierbei ist zu beachten, dass bei der Ausgabe zwischen einer Zeichenkette (String) und dem Array selbst zu unterscheiden ist. Es muss also jedes Mal ein Anführungszeichen gesetzt werden, bevor ein assoziativer Array-Teil ausgegeben wird. Außerdem ist der Punkt (.) nötig, um die Ausgabe auszuführen.

Auch hierbei gibt es wieder die Möglichkeit, das Array mit der Funktion `ARRAY()` zu erzeugen.

Listing 2.10: ■■■
*Assoziatives Array mit
array()*

```

<?php
$schueler = array("JL" => "Jürgen Lange",
                 "AA" => "Azad Adsay");

echo $schueler["JL"];
echo "<br>";
echo $schueler["AA"];

?>

```

Das Besondere hierbei ist der OPERATOR „=>“, der dem Index den Wert zuweist.

Mehrdimensionale Arrays

Wie einige Absätze vorher schon angesprochen, können Arrays mehrere Dimensionen haben, beispielsweise um die Namen der Schüler in Vor- und Zunamen zu unterteilen. In diesem Zusammenhang kommt die Funktion `ARRAY()` zum Tragen.

```

<?php

$schueler = array("schueler1" =>array("Jürgen","Lange"),
                 "schueler2" =>array("Azad","Adsay"),
                 "schueler3" =>array("Martin","Brochier")
                );

// Ausgabe des Namens von Schüler 1
echo "Schüler 1: ", $schueler["schueler1"][0], " ";
echo $schueler["schueler1"][1], "<br>";

// Ausgabe des Namens von Schüler 2
echo "Schüler 2: ", $schueler["schueler2"][0], " ";
echo $schueler["schueler2"][1], "<br>";

// Ausgabe des Namens von Schüler 3
echo "Schüler 3: ", $schueler["schueler3"][0], " ";
echo $schueler["schueler3"][1], "<br>";

?>

```

■ ■ ■ *Listing 2.11:*
**Mehrdimensionales
Array**

Ausgabe:

```

Schüler 1: Jürgen Lange
Schüler 2: Azad Adsay
Schüler 3: Martin Brochier

```

Das Listing wirkt auf den ersten Blick sicherlich ein wenig umständlich und zu ausführlich, aber das Beispiel dient auch nur zur Veranschaulichung, da man normalerweise doch etwas komplexere mehrdimensionale Arrays verwendet.

Wichtig ist nur, dass diesmal statt des Punkts (.) ein Komma (,) gesetzt werden muss und dass die Ausgabe (Auswahl der Namen) über die Schreibweise

```
$array["index_assoziatives_array"]["index_indiziertes_array"]
```

erfolgt. Um beispielsweise den Vornamen des ersten Schülers und den Nachnamen des dritten Schülers zu bekommen, wäre folgende Codezeile nötig:

```

echo "Schüler 3: ", $schueler["schueler1"][0], " ";
echo $schueler["schueler3"][1], "<br>";

```

Arrays leeren

Falls Sie den Inhalt eines großen Arrays einmal während eines Prozesses nicht mehr benötigen sollten, empfehle ich Ihnen, ihn aus Performance-Gründen zu löschen.

Das Löschen eines Arrays ist über die gleiche Methode möglich, über die Sie ein neues Array erzeugen:

```
$leeres_array = array();
```

2.8 Konstanten

Konstanten sind, wie der Name schon sagt, Variablen, deren Inhalt einmalig definiert wird und dann über das Projekt nicht mehr geändert werden kann.

Konstanten werden mit Hilfe der Funktion `DEFINE()` deklariert.

*Listing 2.12: ■■■
Konstanten definieren
und ausgeben*

```
<?php
// Konstante definieren
define("Konstante", 1000);

// Konstante ausgeben
echo(Konstante);

?>
```

Ausgabe:

```
1000
```

Konstanten haben in PHP eine geringere Bedeutung, da sie sehr umständlich zu definieren sind, können aber das eine oder andere Mal ganz nützlich sein.

2.9 Operatoren

Wir haben in den vorherigen Abschnitten bereits unbewusst eine kleine Anzahl von Operatoren verwendet, beispielsweise um Variablen festzulegen oder mehrdimensionale Arrays mit Hilfe der `ARRAY()`-Funktion zu erzeugen.

Zuweisungsoperatoren

Die einfachsten Operatoren sind die Zuweisungsoperatoren. Man benötigt sie beispielsweise, um Variablen einen Wert zuzuweisen. Dazu wird das Gleichheits-Zeichen „`=`“ verwendet.

```

<?php

$neue_zahl = 0;
$zahl      = 1984;

// Zahl wird zu 0
$zahl      = $neue_zahl;

// Zahl, Zahl1, Zahl2, Zahl3 und Zahl 4 werden 1984
$zahl      = $zahl1 = $zahl2 = $zahl3 = $zahl4 = 1984;

?>

```

■ ■ ■ *Listing 2.13:*
Zuweisungsoperator
 „=“

Arithmetische Operatoren

Früher oder später wird es natürlich notwendig, Zahlenwerte zu addieren, zu subtrahieren oder andere Rechenoperationen damit auszuführen. Hier kommen die arithmetischen Operatoren ins Spiel.

```

$zahl + $zahl2; // Addition
$zahl - $zahl2; // Subtraktion
$zahl / $zahl2; // Division
$zahl * $zahl2; // Multiplikation
$zahl % $zahl2; // Modulo (Rest der Ganzzahldivision)

```

Letzteres setzt voraus, dass beide Operanden ganzzahlig sind.

Des Weiteren haben Sie die Möglichkeit, Werte mit einem Befehl um eins zu erhöhen oder zu verringern. Dazu benutzen Sie die Inkrement- und Dekrementoperatoren.

```

$zahl++
$zahl--

```

Hierbei ist es entscheidend, wo die Operatoren angewendet werden, ob vor der Variablen oder danach. Ein Beispiel wird das verdeutlichen.

```

<?php

// Inkrementoperator
$x = $y++; // x wird y, danach wird y um 1 erhöht
$x = ++$y; /* y wird um 1 erhöht, danach wird x der Wert von y
zugewiesen */

// Dekrementoperator
$x = $y--; // x wird y, danach wird y um 1 verringert
$x = --$y; /* y wird um 1 verringert, danach wird x der Wert von
y zugewiesen */

?>

```

■ ■ ■ *Listing 2.14:*
Wirkungsweise des
Inkrement- und
Dekrementoperators

String-Operatoren

Für Zeichenketten gibt es eine wichtige Operation: das Vereinen von Zeichenketten. Das wird, wie ebenfalls bereits angewendet, über den Punkt (.) erreicht.

Listing 2.15: ■■■
Kombination von Zeichenketten

```
<?php
$zeichenkette1 = "PHP und";
$zeichenkette2 = " Flash";

// Vereinigung der beiden Zeichenketten
echo ($zeichenkette1 . $zeichenkette2);

?>
```

Ausgabe:

```
PHP und Flash
```

Wichtig ist hierbei nur, dass nicht die Leerzeichen vor und hinter dem Punkt für den Abstand zwischen den Zeichenketten zuständig sind, sondern das Leerzeichen innerhalb der Zeichenketten!

Logische Operatoren/boolesche Werte

Die booleschen Werte („boolean“, benannt nach dem Mathematiker George Boole) unterscheiden nur zwei Zustände: Ist das Licht an oder ist das Licht aus bzw. allgemein gesagt ist etwas wahr oder falsch (TRUE oder FALSE). Solche Werte kann man auch verknüpfen, um beispielsweise abzufragen, ob ein Wert UND ein anderer Wert wahr sind. Ein zweites Beispiel wäre die Frage, ob der erste Wert NICHT wahr UND der zweite Wert WAHR ist. Für solche Abfragen verwendet man logische Operatoren (siehe Tabelle).

Operator	Bezeichnung	Beispiel	Ergebnis
&& and	Logisches UND	true && false true AND false	false
 or	Logisches ODER	false true false OR true	true
xor	Entweder oder	false xor true	true
!	Negation	!(false)	true

■■■ **Tabelle 2.1: Logische Operatoren/boolesche Werte**

Vergleichsoperatoren

Folgende Operatoren dürften jedem, der sich bereits ein wenig mit Programmierung beschäftigt hat oder sich noch dunkel an seine Schulzeit erinnert, bereits logisch erscheinen.

Vergleichsoperatoren werden am häufigsten für Fallunterscheidungen, wie z.B. eine IF-ELSE-Struktur, benötigt. Dabei werden zwei Werte miteinander verglichen. Und wie in der Grundschule gibt es hier ebenfalls Operatoren wie z.B. GRÖßER ALS, KLEINER ALS usw.

```
<?php

$gewicht = 100;

// Wenn das Gewicht größer 150 ist, dann hast du Übergewicht
if ($Gewicht >= '150') {
    echo "Du hast Übergewicht!";
}

// Alles andere ist im grünen Bereich
else {
    echo "Alles im grünen Bereich!";
}

?>
```

■ ■ Listing 2.16:
Vergleichsoperatoren –
Beispiel

Operator	Bezeichnung	Beispiel	Ergebnis
==	gleich	10 == 10	true
===	identisch (gleicher Datentyp!)	10 === "10"	false
!=	ungleich	10 != 9	true
!==	nicht genau gleich	10 !== "10"	true
>	größer als	10 > 100	false
<	kleiner als	10 < 100	true
>=	größer gleich	10 >= 10	true
<=	kleiner gleich	10 <= 9	false

■ ■ Tabelle 2.2: Vergleichsoperatoren

Ein Anwendungsbeispiel dazu:

```
<?php

if(10 !== "10") {
    echo"Richtig";
}
```

■ ■ Listing 2.17:
10 ist nicht gleich "10"

```

else {
    echo "Falsch";
}

?>

```

2.10 Fallunterscheidungen

Auch Fallunterscheidungen sind bis jetzt schon einige durchgeführt worden. Beispielsweise ist das letzte Listing ebenfalls eine Fallunterscheidung, bei der überprüft wird, ob `10` nicht genau gleich „10“ ist (also inklusive Datentyp). Da dies zutrifft, wird nur der echo-Befehl „richtig“ angezeigt, nicht aber „falsch“.

Fallunterscheidungen sind essentiell für jede Programmiersprache, jeden Programmierer und jedes Programm. Kein anspruchsvolles Programm kommt ohne solche Fallunterscheidungen aus. Sie ermöglichen dynamisches Reagieren auf Benutzereinflüsse und auf sich ändernde Variablenwerte.

„WENN DIES – DANN TUE DAS – ANSONSTEN JENES“

If-Else

If-Else ist die am häufigsten verwendete Fallunterscheidung. Sie erlaubt es, mit Hilfe der gerade kennen gelernten Vergleichsoperatoren, Variablen zu vergleichen und dementsprechende Algorithmen auszuführen. Sie haben unbewusst schon sehr viele If-Else-Fallunterscheidungen in diesem Kapitel gesehen, weshalb ich nur noch kurz auf die einzelnen Strukturen eingehen möchte.

► IF

```

if (bedingung1) {
    anweisung1;
}

```

Wenn `bedingung1` zutrifft, wird die `anweisung1` ausgeführt und auch nur dann!

► ELSE

```

if (bedingung1) {
    anweisung;
}
else {
    anweisung2;
}

```

Wenn bedingung1 zutrifft, wird anweisung1 ausgeführt. Wenn bedingung1 nicht zutrifft, wird anweisung2 ausgeführt!

► **ELSEIF**

```
// Bedingung 1
if (bedingung1) {
    anweisung1;
}

// Bedingung 2
elseif (bedingung2) {
    anweisung2;
}

// Bedingung 3
elseif (bedingung3) {
    anweisung3;
}
// beliebig viele weitere elseif-Blöcke
...
// alles andere
else {
    anweisungN;
}
```

Wenn bedingung1 zutrifft, wird anweisung1 ausgeführt. Wenn bedingung2 zutrifft, wird anweisung2 ausgeführt. Wenn bedingung3 zutrifft, wird anweisung3 ausgeführt usw. Wenn keine der Bedingungen zutrifft, wird anweisungN ausgeführt!

Genauso wie mehrere Else-if-Blöcke nacheinander funktionieren, funktionieren auch beliebig viele If-Blöcke nacheinander. Des Weiteren lassen sich innerhalb eines solchen Blocks, mit Hilfe der logischen Operatoren, mehrere Bedingungen überprüfen. Ein Beispiel dazu:

```
<?php

$zahl1 = 100;
$zahl2 = 200;

if($zahl1 <= $zahl2 AND $zahl1 != $zahl2) {
    echo "Zahl1 ist kleiner gleich Zahl2 und doch ungleich Zahl2";
}

else {
    echo "Zahl1 ist gleich Zahl2";
}

?>
```

■ ■ Listing 2.18:
*Mehrere Bedingungen
in einem If-Block*

Ausgabe:

```
Zahl1 ist kleiner gleich Zahl2 und doch ungleich Zahl2
```

Switch

Es gibt einen Befehl, um das Unterscheiden der Fälle bei vielen möglichen Bedingungen ein wenig zu erleichtern. Die Funktion `switch()` ermöglicht es, einen Wert auf beliebig viele Optionen zu überprüfen, ohne ständig neue Blöcke schreiben zu müssen.

Listing 2.19: ■■■
Ausgabe des
Wochentags mit Hilfe
der `SWITCH()`-Funktion

```
<?php
/* $heute wird mit dem Namen des heutigen Tages auf Englisch
belegt */
$heute = date("l");

echo "$heute<br>";

// mit switch wird überprüft, welcher Tag das ist
// und dieser dann auf Deutsch ausgegeben

switch($heute) {
case "Monday"   :
    echo "Montag";
    break;
case "Tuesday"  :
    echo "Dienstag";
    break;
case "Wednesday":
    echo "Mittwoch";
    break;
case "Thursday" :
    echo "Donnerstag";
    break;
case "Friday"   :
    echo "Freitag";
    break;
case "Saturday" :
    echo "Samstag";
    break;
case "Sunday"   :
    echo "Sonntag";
    break;
}

?>
```

Ausgabe (13.04.02):

```
Saturday
Samstag
```

Hierbei ist zu beachten, dass jede Option mit dem `BREAK;`-Befehl zu beenden ist.

Denselben Effekt hätte man natürlich auch mit If-Strukturen erreichen können. Nur der Anschaulichkeit halber dazu noch ein Beispiel:

```
<?php

// $heute wird mit dem Namen des heutigen Tages auf Englisch
belegt
$heute = date("l");

// Heutigen Tag in Englisch ausgeben
echo "$heute<br>";

// Welcher Tag ist heute?
if ($heute == "Monday") {
    echo "Montag";
}
if ($heute == "Tuesday") {
    echo "Dienstag";
}
if ($heute == "Wednesday") {
    echo "Mittwoch";
}
if ($heute == "Thursday") {
    echo "Donnerstag";
}
if ($heute == "Friday") {
    echo "Freitag";
}
if ($heute == "Saturday") {
    echo "Samstag";
}
if ($heute == "Sunday") {
    echo "Sonntag";
}

?>
```

■ ■ ■ *Listing 2.20:*
Gleicher Effekt mit
If-Strukturen

Ausgabe (13.04.02):

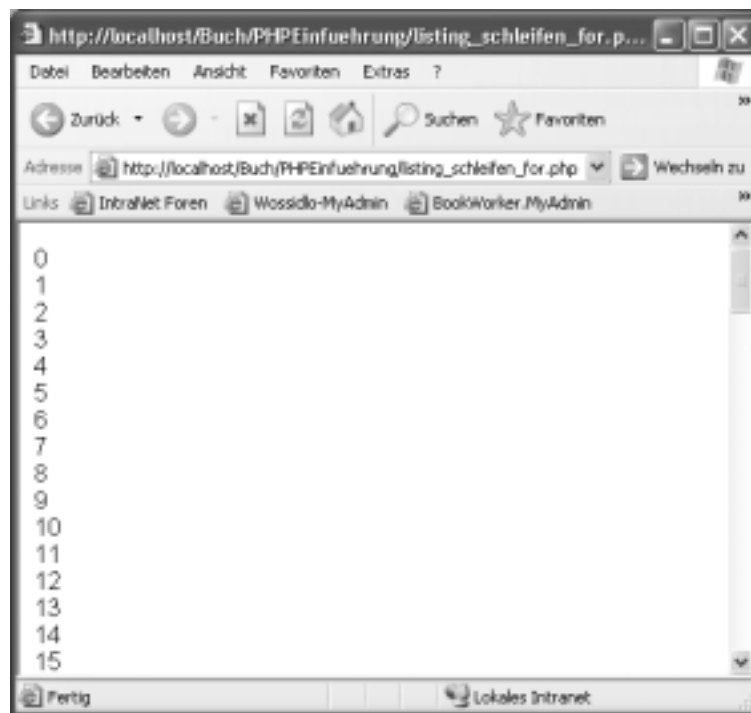
```
Saturday
Samstag
```

Dieses Beispiel ist weitaus umständlicher als bei Verwendung der SWITCH()-Funktion.

2.11 Schleifen

Schleifen sind ebenfalls eine wichtige Grundlage für professionelles Programmieren. Wie die Fallunterscheidungen haben auch die Schleifen wieder mit Bedingungen zu tun, nur dass sie nicht nur überprüfen, ob eine Bedingung erfüllt ist oder nicht, sondern auch dazu beitragen, dass eine Bedingung erfüllt bzw. nicht mehr erfüllt wird. Schleifen lassen bestimmte Anweisungen so oft „durchlaufen“, bis eine Bedingung erfüllt bzw. wiederum nicht mehr erfüllt ist.

For-Schleifen



■ **Abbildung 2.9:** Ausgabe einer Zahlenfolge mit Hilfe einer For-Schleife

Die For-Schleife ist eine der grundlegendsten Schleifen in PHP. Sie kann zum Beispiel dazu genutzt werden, eine Anweisung genau 100 Mal durchlaufen zu lassen. Die Syntax der For-Schleife sieht wie folgt aus:

```
For(Variable = Anfangswert, Bedingung, Durchlauf) {  
    Anweisung;  
}
```

ANFANGSWERT (INIT): die Anweisung, in der geklärt wird, von wo aus die Variable hochgezählt werden soll

BEDINGUNG: die Bedingung, die nach jedem Durchlauf auf Gültigkeit geprüft wird

DURCHLAUF: meist Inkrement- oder Dekrementoperator (siehe arithmetische Operatoren)

Deutlicher wird das Ganze anhand eines Praxisbeispiels. Bleiben wir doch gleich bei den schon angesprochenen 100 Durchläufen:

```
<?php
for($i = 0; $i <= 100; $i++) {
    echo "$i<br>";
}

?>
```

■ ■ ■ Listing 2.21:
For-Schleife

Ausgabe:

```
1
2
3
...
```

While-Schleifen

While-Schleifen ähneln meiner Meinung nach sehr If-Else-Strukturen, da sie einen Block auch nur so lange ausführen, solange die Bedingung erfüllt ist. Aus der Syntax

```
While (Bedingung) {
    Anweisung;
}
```

lässt sich entnehmen, dass eine While-Schleife auch nicht unmittelbar (wie z.B. die For-Schleife) in die Bedingung eingreift, d.h. die Bedingungsvariable erhöht oder verringert. Das sollte immer innerhalb der Anweisung geschehen, wie Sie später in einem Beispiel sehen werden. Die Gefahr von so genannten Endlos-Schleifen ist hier sehr hoch.

Endlos-Schleifen führen zu enormer Prozessorbelastung und bringen meistens nichts als Ärger; versuchen Sie also, so etwas zu vermeiden.

Eine While-Schleife hat noch einen Vorteil. Sie muss nicht zwangsläufig einmal durchlaufen werden. Sollte die Bedingung nicht mehr erfüllt sein, wenn der PHP-Parser an die Stelle kommt, wo eine While-Schleife genau diese Bedingung verlangt, dann wird sie gnadenlos übergangen. Sie nimmt somit auch die Funktion einer Sicherungsschleife ein. Wenn Sie also ab einem bestimmten Punkt in Ihrem Programm eine Variable mit genau diesem Wert benötigen, kann eine While-Schleife Ihnen helfen. Die Anwendungsmöglichkeiten sind da wirklich komplex.

Aber nun zu einem Beispiel:

Listing 2.22: ■■■
While-Schleife

```
<?php
$i = 100;

while ($i >= 0) {

    // $i ausgeben, Zeilenumbruch
    echo "$i<br>";

    // $i verringern
    $i--;

}
?>
```

Ausgabe:

```
100
99
98
...
```

Do-While-Schleifen

Die Do-While-Schleife unterscheidet sich unter den vorliegenden Gesichtspunkten nur in einer Sache von der normalen While-Schleife: Ihre Bedingung wird erst am Ende des Blocks überprüft. Genauer gesagt heißt das, dass diese Schleife (wie z.B. die Repeat-Until-Schleife bei Objekt-Pascal – Delphi) mindestens einmal durchlaufen wird. Die Syntax lautet wie folgt:

Listing 2.23: ■■■
Do-While-Schleife

```
Do {
    Anweisung;
} while (Bedingung);
```

Dazu ebenfalls ein Beispiel:

```
<?php
// Zahl definieren
$zahl = 0;

// Schleifenbeginn
do {
    // Zahl um eins erhöhen und ausgeben (mit Umbruch)
    $zahl = $zahl + 10;
    echo"$zahl<br>";
```

```

    // Schleifenende
}
//Bedingung
while ($zahl < 1000);

?>

```

2.12 Funktionen

Funktionen sind eines der interessantesten Themen von PHP. Sie sind zwar im Grunde genommen nichts anderes als eine Möglichkeit, um Ihren Code gut zu strukturieren, eröffnen aber auf der anderen Seite ungeahnte Möglichkeiten für Ihre PHP-Programmierung.

Wie schon gesagt, ist eine Funktion eine Art Code-Speicher, den Sie immer wieder in verschiedenen Teilen Ihres Programms verwenden können. Das erspart erstens das Debugging und zweitens sehr viel Schreibarbeit.

Die Syntax einer Funktion lautet wie folgt:

```

function Funktionsname([Parameter]) {
    [Kommandos]
    [return $Rückgabewert;]
}

```

Das heißt im Prinzip nichts anderes, als dass Sie die Möglichkeit haben, eine Standardroutine mit einem Parameter, also einem variablen Wert, zu füttern, um am Ende einen Rückgabewert, wiederum eine Variable, zu erhalten. In der Mathematik dürfte Ihnen jetzt der Ausdruck $f(x)$ einfallen. Und Sie haben Recht. Es ist nichts anderes.

Neben den unendlich vielen Standardfunktionen, die Ihnen PHP bereits „mitliefert“ (darunter u.a. Datenbankfunktionen, FTP-Funktionen, String- oder Array-Funktionen), können Sie natürlich auch unendlich viele eigene Funktionen schreiben. Das möchte ich Ihnen anhand eines Beispiels erläutern.

Eigene Funktionen schreiben

Für unser Beispiel denken wir uns den imaginären Problemfall, dass man eine Funktion benötigt, die eine beliebige Zahl quadriert. Das ist sicher kein schwerwiegendes Problem, aber es veranschaulicht sehr gut die Funktionsweise von Funktionen.

Listing 2.24: ■■■
**Eigene Funktionen
 schreiben**

```
<?php
// Funktion zum Quadrieren einer Zahl schreiben
// Funktionsname: QuadratErrechnen

function QuadratErrechnen($zahl) {
    $ergebnis = $zahl * $zahl;

    // Wert an Code zurückgeben
    // Achtung zurückgeben ist nicht gleich ausgeben!
    return $ergebnis;
}

// Ende Funktion QuadratErrechnen
$meinezahl = 10;

// $meinezahl an Funktion übergeben
$meinergebnis = QuadratErrechnen($meinezahl);

// Ergebnis ausgeben
echo "$meinergebnis";

?>
```

Ausgabe:

100

Beachten Sie bitte: Sobald Sie den RETURN-Befehl verwenden, wird die Funktion abgebrochen und der Wert an den Code übergeben. Folgendes Beispiel würde also nicht wie erhofft funktionieren:

Listing 2.25: ■■■
Fehlerhafte Funktion

```
<?php
function Fehlerhaft($zahl) {
    $ergebnis = $zahl * $zahl;

    return $ergebnis;

    $ergebnis = $ergebnis +10;

    return $ergebnis;
}

$zahl = 10;

$ausgabe = Fehlerhaft($zahl);

echo"$ausgabe";

?>
```

Ausgabe:

```
100
```

Hier wird das Ergebnis 100 ausgegeben, nicht das erwartete Ergebnis von 110.

Parameter

Sie können an Funktionen Parameter übergeben, die innerhalb des Funktionsblocks Verwendung finden. Schreiben Sie sie dazu einfach innerhalb der runden Klammern, wie in dem folgenden Beispiel. Mehrere Parameter werden per Komma getrennt.

```
<?php
function Addition($zahl1,$zahl2) {
    $ergebnis = $zahl1 + $zahl2;
    return $ergebnis;
}

$meinezahl1 = 2000;
$meinezahl2 = 5000;

$meinergebnis = Addition($meinezahl1,$meinezahl2);

echo "$meinergebnis";

?>
```

■ ■ ■ *Listing 2.26:*
Mehrere Parameter in einer Funktion

Ausgabe:

```
7000
```

In die runden Klammern werden alle benötigten Parameter einer Funktion geschrieben. Es gibt allerdings auch optionale Parameter einer Funktion, die nicht unbedingt übergeben werden müssen. Dazu habe ich eine Funktion vorbereitet, die Ihren Text fett schreibt und eine Fehlermeldung ausgibt, wenn Sie keinen Text angegeben haben.

```
<?php
function Fett($text = "Kein Text angegeben !") {
    echo "<b>$text</b>";
}
}
```

■ ■ ■ *Listing 2.27:*
Funktion FETT()

```

    $fettertext = Fett();
    echo "$fettertext";
?>

```

Ausgabe:

Kein Text angegeben !

Wenn Sie jedoch einen Text angeben, wird dieser automatisch fett geschrieben:

Listing 2.28: ■■■
Funktion FETT() zum
Zweiten

```

<?php
function Fett($text = "Kein Text angegeben !") {
    echo "<b>$text</b>";
}

$text = "Das ist mein fett geschriebener Text!";
$fettertext = Fett($text);
echo "$fettertext";
?>

```

Ausgabe:

Das ist mein fett geschriebener Text!

PHP-Funktionen

PHP beinhaltet von vornherein eine riesige Bibliothek mit Stammfunktionen für die unterschiedlichsten Zwecke. Um einen kurzen Überblick zu bekommen, wofür PHP bereits Funktionen bereithält, hier eine kurze Auflistung.

Funktionsart	Beispielfunktionen (Auswahl)
Variablenfunktionen	define(), global(), gettype()
Array-Funktionen	array(), asort(), current(), next(), shuffle(), array_count()
Zeichenkettenfunktionen	addslashes(), explode(), md5(), soundex(), htmlspecialchars(), rawurlencode()
Rechtschreibfunktionen	pspell_new(), pspell_suggest()
Funktionen für reguläre Ausdrücke	ereg(), ereg_replace(), split()
Datums-, Kalender- und Zeitfunktionen	checkdate(), date(), localtime(), time(), getdate(), jdtojulyan(), juliantojd()
Mathematische Funktionen	abs(), cos(), rand(), round(), sqrt(), tan(), sin()
Dateifunktionen	dir(), closedir(), chmod(), dirname(), fopen(), fclose(), diskfreespace(), file_exists(), is_dir(), popen()
Session-Funktionen	session_destroy(), session_id(), session_start(), session_register(), session_name(), session_encode()
Netzwerkfunktionen	fsockopen(), long2ip(), checkdnsrr(), gethostbyaddr()
FTP-Funktionen	ftp_connect(), ftp_login(), ftp_mkdir(), ftp_pwd(), ftp_rename(), ftp_delete(), ftp_quit()
Verschlüsselungsfunktionen	mcrypt_get_key_size()
XML-Funktionen	xml_parser_create(), xml_set_element_handler()
PDF-Funktionen	pdf_get_info(), pdf_open(), pdf_close(), pdf_begin_page(), pdf_circle(), pdf_rotate()
Shockwave-Flash-Funktionen	swf_openfile(), swf_getframe(), swf_placeobject(), swf_fontsize(), swf_rotate(), swf_definetext()
Mail-Funktionen	mail(), imap_append(), imap_createmailbox(), imap_header()
Bildfunktionen	getimagesize(), imagearc(), imagecreate(), imagefontheight(), imagecreatefromjpeg(), imagecreatefrompng(), imagecreatefromgif()
Datenbankfunktionen	dbase_create(), dbase_open(), MySQL_connect(), MySQL_fetch_array(), MySQL_query(), msql(), odbc_do(), odbc_connect(), ocilogon(), ora_error(), pg_connect()

■ **Tabelle 2.3: PHPs große Vielfalt an Stammfunktionen (Auswahl des Autors)**

Diese und viele weitere Funktionen, ihren Zweck und ihre Funktionsweise finden Sie im offiziellen PHP-Manual unter <http://www.php.net>.

2.13 Datumsprobleme

Um einmal die Anwendung dieser Funktionen in der Praxis zu zeigen, habe ich das Thema Daten und Zeiten gewählt, da Ihnen dieses Gebiet sehr häufig in der Programmierung begegnen wird.

Das aktuelle Datum in verschiedenen Schreibweisen bekommt man ganz einfach mit der Funktion `DATE()`. Die Funktion `DATE()` benötigt wiederum einige Parameter, die festlegen, in welchem Format das Datum ausgegeben werden soll. Dazu folgende Übersicht:

Parameter	Erläuterung
a	für „am“ und „pm“
A	für „AM“ und „PM“
d	für Tag des Monats mit zwei Stellen und führender Null: „01“ bis „31“
D	für Tag der Woche mit Abkürzung mit drei Buchstaben: „Sun“ usw.
F	für Monat, ausgeschrieben: „June“
h	für Stunde im 12-Stunden-Format: „01“ bis „12“
H	für Stunde im 24-Stunden-Format: „00“ bis „23“
g	für Stunde im 12-Stunden-Format ohne führende Null: „1“ bis „12“
G	für Stunde im 24-Stunden-Format ohne führende Null: „0“ bis „23“
i	für Minuten: „00“ bis „59“
j	für Tag des Monats ohne führende Null: „1“ bis „31“
l (kleines „L“)	für Wochentag voll ausgeschrieben: „Sunday“
L	für boolescher Wert, ob Schaltjahr oder nicht: „0“ oder „1“
m	für Monat mit führender Null: „01“ bis „12“
n	für Monat ohne führende Null: „1“ bis „12“
M	für Monat als Abkürzung: „Jan“
s	für Sekunden mit führender Null: „00“ bis „59“
S	für das Suffix englischer Zahlen: „th“, „nd“ oder „st“ ...
t	für Anzahl der Tage in einem Monat: „28“ bis „31“
U	für Sekunden seit Beginn der Unix-Epoche am 1.1.1970
w	für numerische Darstellung des Wochentags: „0“ (Sonntag) bis „6“ (Samstag)
Y	für vierstelliges Jahr: „2002“
y	für zweistelliges Jahr: „02“
z	für Tag im Jahr: „0“ bis „356“
Z	für die Differenz zur Zeitzone in Sekunden: „-43200“ bis „43200“ (12 Stunden)

■ ■ ■ **Tabelle 2.4: Parameter der Funktion `DATE()`**

Quelle: <http://www.php.net/manual/de/function.date.php>

Diese Parameter werden dann in einem Skript in die `DATE()`-Funktion eingesetzt und liefern die entsprechenden Werte zurück. Dazu verwende ich noch einmal das erste Listing aus diesem Kapitel:

```
<?php

// Aktuellen Tag - numerisch - ermitteln
$tag = date("w");

// per Fallunterscheidung ins Deutsche übersetzen und ausgeben
switch ($tag) {

    case 7: echo "Sonntag";
           break;
    case 0: echo "Sonntag";
           break;
    case 1: echo "Montag";
           break;
    case 2: echo "Dienstag";
           break;
    case 3: echo "Mittwoch";
           break;
    case 4: echo "Donnerstag" ;
           break;
    case 5: echo "Freitag";
           break;
    case 6: echo "Samstag" ;
           break;

    default: echo "komischer Tag";
}

// Datum im Format TT.MM.YYYY nach Komma ausgeben
echo date(", d.m.Y ");

?>
```

Listing 2.29:
Die DATE()-Funktion in Aktion

Ausgabe (am 14.04.2002):

```
Sonntag, 14.04.2002
```

