PHP - Sprachelemente und Syntax

In diesem Skript werden die wesentlichen Bestandteile und Strukturen der PHP-Programmierung vorgestellt. Da diese in allen Programmiersprachen in ähnlicher Form vorhanden sind, können Leser mit ausreichender Programmiererfahrung das Skript gerne überspringen, ohne den roten Faden zu verlieren. Dies ist aber nicht anzuraten, da doch immer wieder mit kleineren und größeren Abweichungen zwischen Programmiersprachen zu rechnen ist. Selbst erfahrenen Programmierern wird empfohlen, das Skript zumindest zu überfliegen.

1.1 Integration von PHP

Hier nochmals eine kurze Zusammenfassung der Schreibweisen.

SGML-Stil (Standard Generalized Markup Language)

```
<?
        echo "Einbindung in SGML-Stil";
?>
```

XML (Extensible Markup Language)

```
<?PHP
        echo "Einbindung in XML-Stil";
?>
```

```
<?php
        echo "Einbindung in XML-Stil";
?>
```

ASP-Stil (Active Server Pages)

```
<%
        echo "Einbindung in ASP-Stil";
%>
```

<script>-Tag

```
<script language="php">
        echo "Einbindung im JS-Stil";
</script>
```

1.1.1 Notationshinweise

Am Anfang ist die Gefahr groß, unübersichtliche Skripts zu erstellen, da Leerzeichen, Zeilenumbrüche und Tabulatoren vom PHP-Interpreter ignoriert werden. Sie können eine HTML-Seite samt PHP-Bestandteil wie folgt schreiben:

```
<html>
<head><title>Erste Schritte</title></head>
<body><?php echo "Dies ist ein Test" ?>
</body>
</html>
```

aber auch so:

```
<html>
<head>
       <title>Erste Schritte</title>
</head>
<body>
<?php
       echo "Dies ist ein Test"
?>
</body>
</html>
```

Je nach Umfang des in PHP geschriebenen Codes wird die eine oder andere Variante günstiger sein. Eine optisch eindeutige Trennung von HTML und PHP hat sich in der Praxis als sinnvoll erwiesen. Im nächsten Abschnitt wird das Einschließen von Dateien vorgestellt. Sie sollten so oft wie möglich Ihre PHP-Bestandteile auslagern. Diese Dateien werden dann Module genannt. Sie können so den Code in der HTML-Seite überschaubar halten. Bei größeren Anwendungen vereinfacht sich die Wartung deutlich.

Achtung: Befehle werden in PHP generell mit einem Semikolon abgeschlossen, lediglich wenn der Befehl allein zwischen <? und ?> steht, ist dies optional.

1.1.2 Einbindung externer Skripts

Include

Der Befehl `include("dateiname");` fügt an dieser Stelle den Inhalt der Datei *dateiname* ein. Dadurch ist es möglich, Quellcode, der in mehreren Dateien benötigt wird, zentral zu halten, so dass Änderungen einfacher durchzuführen sind.

Die Datei, die eingefügt wird, wird als HTML-Code interpretiert. Deshalb muss, wenn in der Datei nur PHP-Code steht, diese Datei mit `<?php anfangen und mit ?>` enden bzw. mit anderen PHP-Code-Markierungen. Wenn `include()` in Verbindung mit Bedingungen oder Schleifen eingesetzt wird, muss es immer in geschweiften Klammern geschrieben werden.

```
// Falsch
if ( $Bedingung )
include ("Datei.inc");
```

```
// Richtig
if ( $Bedingung ) {
include ("Datei.inc");
}
```

require

Ganz analog zu `include()` funktioniert `require()`. Es wird aber von PHP etwas anders behandelt. Der `require()`-Ausdruck wird beim ersten Aufruf durch die Datei ersetzt. Wie bei `include()` wird erst einmal aus dem PHP-Modus herausgegangen. Es gibt drei wesentliche Unterschiede zu `include()`:

- Zum einen wird `require()` immer ausgeführt, also auch dann, wenn es eigentlich abhängig von einer `IF`-Anweisung nicht ausgeführt werden dürfte.
- Zum anderen wird es innerhalb einer Schleife (for, while) nur ein einziges Mal ausgeführt – egal, wie oft die Schleife durchlaufen wird, der Inhalt der eingefügten Datei wird mehrmals abgearbeitet.
- Zum Dritten liegt der Unterschied in der Reaktion auf nicht vorhandene Dateien: `include()` gibt nur ein »Warning« aus und PHP läuft weiter, bei `require()` bricht PHP mit einem »Fatal error:« ab.

```
// Schreibweisen
require 'funktionen.php';
require $datn;
require ('funktionen.txt');
```

include_once, require_once

Seit PHP4 gibt es neben den Funktionen `include()` und `require()` auch noch die Funktionen `include_once()` und `require_once()`. Der Name zeigt schon, wo der Unterschied liegt: Bei den `*_once()`-Funktionen wird die Datei nur einmal eingefügt, unabhängig davon, wie häufig man versucht, sie einzufügen.

Der Sinn ist einfach: Bei umfangreichen Webseiten gibt es häufig eine Datei, die die zentralen Funktionen enthält. Da diese in den Webseiten benötigt werden, fügt man die Datei immer am Anfang ein. So weit kein Problem. Sobald aber mehrere zentrale Funktionsdateien existieren, die sich auch untereinander benötigen, wird es schwierig, weil jede nur einmal eingefügt werden darf.

Tipps im Umgang mit externen Dateien

Normalerweise können Benutzer den Inhalt der Skripts nicht sehen. Jede Datei mit der Endung .php3, .php4 oder .php wird vom Webserver an den PHP-Interpreter weitergeleitet und von diesem verarbeitet. Es ist natürlich ohne weiteres möglich, jede andere Endung anzugeben. Oft werden Dateien, die mit `include` eingeschlossen werden sollen, mit .inc bezeichnet. Auch diese Endung wird nicht verarbeitet. Das ist für den Ablauf

des Skripts egal – die Verarbeitung erfolgt im Rahmen des »umgebenden« Skripts und damit unter dessen Regie.

Was jedoch nicht egal sein dürfte, ist das damit aufkommende Sicherheitsproblem. Sollte ein Nutzer den Pfad zu den Include-Dateien herausbekommen, kann er deren Namen in der Adresszeile des Browsers direkt eingeben. Der Webserver wird die Endung nicht kennen und dem Browser die Datei direkt zusenden. Der Browser erkennt ein einfaches Textdokument und stellt es dar. Da in Include-Dateien auch Kennwörter für Datenbanken stehen können, wäre dies äußerst problematisch.

Dieses Problem kann man jedoch recht schnell beseitigen. Benennen Sie sämtliche Include-Dateien in *.inc.php* um. So haben Sie eine eindeutige Kennzeichnung und erzwingen im Notfall das Parsen des Codes durch den PHP-Interpreter. Das mag zwar zu einer Fehlermeldung führen, sollte diese Datei einzeln aufgerufen werden, an den Inhalt gelangt der Benutzer dennoch nicht.

1.1.3 PHP und JavaScript

Oft wird von der Kombination PHP/JavaScript gesprochen. Beides hat direkt nichts miteinander zu tun. JavaScript wird im Browser abgearbeitet und PHP auf dem Server. Beide Sprachen basieren auf dem ECAMScript 3-Standard, daher sind eine Vielzahl von Sprachbestandteilen in ihrer Anwendung und Syntax identisch. Betrachten Sie JavaSkript als Erweiterung zu HTML. Der neueste Entwicklungsstand DHTML geht ohnehin von JavaScript als Anweisungssprache aus.

Selbstverständlich können Sie ebenso wie HTML auch JavaScript-Anweisungen dynamisch erzeugen oder mit variablen Attributen versehen. Es bleibt Ihnen überlassen, was Sie daraus machen.

Wir werden im Übrigen nicht näher auf JavaScript eingehen, Sie können uns glauben: PHP ist spannend genug.

1.2 Einführung in PHP

Der folgende Abschnitt wendet sich vor allem an die PHP-Entwickler, die für die tägliche Arbeit eine kompakte Referenz benötigen, die sich zum Nachschlagen äußerst gut eignet.

1.2.1 Ausdrücke

Sollte man PHP mit einem Satz charakterisieren wollen, könnte man sagen, dass es sich um eine ausdrucksorientierte Sprache handelt.

Dadurch stellt sich gleich zu Begin die Frage, was ein Ausdruck ist. Ein Ausdruck ist ganz allgemein eine Aneinanderreihung von Zeichen unter Beachtung einer vorgegebenen Syntax. Ausdrücke können ganz unterschiedlich aufgebaut sein. Das wichtigste

Charakteristikum von Ausdrücken ist, dass sie immer einen Wert – und sei es den Wert 0 oder "" (leer String) – besitzen.

Ausdrücke stellen somit die wichtigsten Komponenten von PHP dar.

Elementare Ausdrücke

Der folgende Ausdruck ist im vorstehenden Sinne ein gültiger, elementarer PHP-Ausdruck.

```
1000
```

Es handelt sich um eine Integer-Konstante mit dem Wert 1000. Weitere elementare Ausdrücke sind beispielsweise Konstanten und Variablen.

Zusammengesetzte Ausdrücke

Zusammengesetzte Ausdrücke entstehen dadurch, dass elementare Ausdrücke mithilfe von Operatoren verknüpft oder dass Werte von Ausdrücken mithilfe von Zuweisungsoperatoren anderen Ausdrücken zugewiesen werden.

In der folgenden Anweisung wird dem Ausdruck `$zahl` der Ausdruck 1000, d.h. eine Integer-Konstante mit dem Wert 1000 zugewiesen.

```
$zahl = 1000;
```

Nach dieser Zuweisung ist der Wert von `$zahl` ebenfalls 1000. Somit sind hier zwei Werte im Spiel:

- Der Wert der Integer-Konstanten, nämlich 1000.
- Der Wert von `$zahl`, der auf 1000 geändert wird.

In der folgenden Anweisung wird dem Ausdruck `$punkte` der Ausdruck `$zahl` zugewiesen.

```
$punkte = $zahl;
```

Der gesamte Ausdruck, also `$punkte = $zahl`, hat aufgrund des vorhergehenden Ausdrucks nun den Wert 1000. `$punkte` ist also ebenfalls ein Ausdruck mit dem Wert 1000. Der Ausdruck `$punkte = $zahl` ist dabei gleichbedeutend mit dem Ausdruck `$punkte = $zahl = 1000`.

Funktionen als Ausdrücke

Ein weiteres Beispiel für Ausdrücke sind Funktionen. Funktionen sind ebenfalls Ausdrücke mit dem Wert ihres Rückgabewertes. Die folgende Funktion `wert()` ist also ein Ausdruck mit dem Wert 1000.

```
<?php
function wert() {
       return 1000;
}
```

```
// Ausgabe (1000)
echo wert();
?>
```

Bei dem zusammengesetzten Ausdruck:

```
$resultat = wert();
```

handelt es sich somit ebenfalls um einen Ausdruck mit dem Wert 1000.

Prä- und Post-Inkrement in Ausdrücken

Komplexere Ausdrücke in PHP verwenden die von der Sprache C bekannten Prä- und Post-Inkremente sowie die entsprechenden Dekremente.

Sowohl Prä-Inkremente als auch Post-Inkremente erhöhen den Wert einer Variablen. Der Unterschied besteht im Wert des Inkrement-Ausdrucks:

- Das Prä-Inkrement, welches `++$var` geschrieben wird, enthält als Wert den Wert der erhöhten Variablen.
- Das Post-Inkrement, welches `$var++` geschrieben wird, enthält dagegen den ursprünglichen Wert der Variablen vor der Erhöhung, d.h., PHP erhöht den Wert der Variablen erst, nachdem es ihren Wert ausgelesen hat.

Beispiel Prä-Inkrement:

```
<?php
$zahl = 1000;
echo ++$zahl;
?>
```

Ausgabe:

```
1001
```

Beispiel Post-Inkrement:

```
<?php
$zahl = 1000;
echo $zahl++;
?>
```

Ausgabe:

```
1000
```

Wann ist ein Ausdruck wahr?

Oft ist man nicht am spezifischen Wert eines Ausdrucks interessiert, sondern bewertet lediglich, ob der Ausdruck wahr oder falsch ist.

PHP kennt die Booleschen Konstanten `TRUE (1)` und `FALSE (0)`. Ein Ausdruck ist in PHP dann wahr, wenn ihm, wie im folgenden Beispiel, die Boolesche Konstante `TRUE` oder ein anderer Ausdruck, dessen Wert `TRUE` ist, zugewiesen wurde.

```
<?php
$signal = TRUE;
echo "$signal";
?>
```

Ausgabe:

```
1
```

Vergleichsausdrücke

Eine weitere, auf dem im vorigen Abschnitt eingeführten Wahrheitswert basierende Kategorie von Ausdrücken sind die Vergleichsausdrücke. Vergleichsausdrücke werden z.B. in bedingten Anweisungen unter Verwendung von Vergleichsoperatoren eingesetzt:

```
if ($signal == TRUE) Anweisung;
```

In Vergleichsausdrücken wird immer der Wert zweier Teilausdrücke verglichen. Der Wert des Gesamtausdrucks ist, abhängig vom Ergebnis des Vergleichs, entweder also *Falsch* (0) oder *Wahr* (1).

Kombinierte Zuweisungs- und Operator-Ausdrücke

Sollten Sie schon mit der Sprache C gearbeitet haben, kennen Sie die Möglichkeit, Zuweisungs- und Operator-Ausdrücke zu kombinieren. In PHP ist dies ebenfalls möglich.

Um zum Beispiel den Wert einer Variablen um 10 zu erhöhen, kann in einer Anweisung der folgende Ausdruck verwendet werden:

```
$zahl += 100
```

Das ist gleichbedeutend mit:

»Nimm den Wert von `$zahl`, addiere 100 hinzu und weise den entstandenen Wert der Variablen `$zahl` zu«.

In solchen kombinierten Zuweisungs- und Operator-Ausdrücken kann jeder Operator, der zwei Elemente verbindet, zusammen mit einem Zuweisungsoperator verwendet werden.

```
<?php
$zahl = 100;
echo $zahl *= 10;
?>
```

Ausgabe:

```
1000
```

Konditionale Operatoren in Ausdrücken

Ein weiterer Typ von Ausdrücken, der in PHP oft gebraucht wird und den Sie vielleicht von der Sprache C her kennen, ist der dreifache konditionale Operator:

```
Ausdruck ? Ausdruck2 : Ausdruck3
```

Wenn der Wert des ersten Ausdrucks *Wahr* ist, dann wird der Wert des zweiten Ausdrucks zurückgegeben. Andernfalls, d.h. wenn der Wert von Ausdruck1 *Falsch* ist, nimmt der Wert des Gesamtausdrucks den Wert des dritten Ausdrucks an.

Beispiel:

```
<?php
$punkte = 1000;
$highscore = 500;
$resultat = ($highscore > $punkte ) ? "Alter" : "Neu";
echo $resultat;
?>
```

Ausgabe:

```
Neu
```

Beispiel:

```
<?php
$punkte = 500;
$highscore = 1000;
$resultat = ($highscore > $punkte ) ? "Alter" : "Neu";
echo $resultat;
?>
```

Ausgabe:

```
Alter
```

1.2.2 Anweisungen

Anweisungen werden zur Laufzeit eines Programms abgearbeitet und bewirken in der Regel Änderungen an Datenobjekten oder Interaktionen mit der Programmumgebung.

PHP kennt folgende Anweisungen:

- Zuweisungen
- Funktionsaufrufe
- Schleifen
- Bedingungen

Anweisungen werden in PHP wie in der Programmiersprache C mit einem Semikolon beendet.

Beispiel:

```
<?php
echo ("Hallo Welt!");
?>
```

Ausgabe:

```
Hallo Welt!
```

Nachdem Sie genau wissen, was ein Ausdruck ist, fällt die Definition des Begriffs »Anweisung« nicht mehr schwer. Eine Anweisung ist ein Ausdruck, gefolgt von einem Semikolon, und hat somit die Form:

```
Ausdruck ;
```

Gültige Anweisungen

```
<?php
$vorname = "Gülten";
echo $vorname;
?>
```

1.2.3 Codezeile

Eine Codezeile in PHP muss immer mit einem Semikolon beendet werden. Diese Schreibweise hat sich bei den nach der ECMA-Spezifikation genormten Programmiersprachen durchgesetzt, zu denen unter anderem auch JavaScript gehört.

Beispiel:

```
<?php
// Array - Codezeilen mit Semikolon
$personen = array();
$personen[0] = "Caroline";
$personen[1] = "Matthias";

// Ausgabe - Matthias
echo $personen[1];
?>
```

1.2.4 Semikola

Semikola legen in PHP das Ende einer Anweisung fest. Das basiert auf Programmiersprachen wie C/C++ oder Java. Zusätzlich dient es als Abgrenzung der Anweisungen untereinander.

Beispiel:

```
<?php
// Schreibweise - Fehlerfrei
$vorname = "Caroline";
```

```
$nachnName = "Kannengiesser";

// Schreibweise - Fehlerhaft
$vorname = "Caroline"
$nachname = "Kannengiesser"
?>
```

Sie sollten möglichst auf das Setzen der Semikola achten.

Achtung: Anweisungsblöcke oder Kontrollstrukturen werden nicht in jeder Zeile mit einem Semikolon abgeschlossen, besonders dann nicht, wenn bereits ein Block Trennzeichen, wie zum Beispiel eine geschweifte Klammern, existiert.

Beispiel:

```
<?php

$signal = true;

if ($signal == true) {;
        echo "Signal ist true";
} else {;
        echo "Signal ist false";
};
?>
```

Das Beispiel ist syntaktisch gesehen fehlerfrei und wird auch korrekt ausgeführt. Nur ist es etwas zu viel des Guten, folgende Schreibweise wäre zu empfehlen.

```
<?php

$signal = true;

if ($signal == true) {
        echo "Signal ist true";
} else {
        echo "Signal ist false";
}
?>
```

Beispiel:

```
<?php
for ($i = 0; $i <= 10; $i++) {;
        echo "Wert: " . $i . "<br>";
};
?>
```

Besser:

```
<?php
for ($i = 0; $i <= 10; $i++) {
        echo "Wert: " . $i . "<br>";
}
?>
```

Wie Sie sehen, veranschaulichen diese beiden Fallbeispiele die korrekte Platzierung des Semikolons in einem Anweisungsblock. Bei der Definition einer Funktion kann am Ende ein Semikolon gesetzt werden, das ist jedoch optional.

Beispiel:

```
<?php
// Definition
function addition($zahlEins,$zahlZwei) {
        return $zahlEins + $zahlZwei;
};

// Aufruf der Funktion
$resultat = addition(10,5);

// Ausgabe
echo $resultat;
?>
```

1.2.5 Leerzeichen

PHP ignoriert Leerzeichen, Tabulatoren und Zeilentrenner, solange die im Programm enthaltenen Schlüsselwörter, Bezeichner, Zahlen und andere Einheiten nicht durch ein Leerzeichen oder einen Zeilenumbruch getrennt werden.

Beispiel:

```
<?php
function setze
Ausgabe ($parameter wert) {
        echo $par ameter wert;
}
setzeAusgabe("Ausgabe bitte!");
?>
```

Dieses Fallbeispiel wäre vielleicht, unter dem Aspekt künstlerischer Freiheit betrachtet, ein schönes Bild, syntaktisch jedoch leider eine Katastrophe. Genau dies sollte man vermeiden.

Beispiel:

```
<?php
function setzeAusgabe ($parameterwert) {
        echo $parameterwert;
}
setzeAusgabe("Ausgabe bitte!");
?>
```

So sollte der PHP-Code aussehen. Es bleibt Ihnen überlassen, Leerzeichen, Tabs und Zeilentrenner zu verwenden, doch sollten diese die Syntaxregeln beachten, dann steht einer optimalen Formatierung des Code nichts im Weg und die Programme sind leicht lesbar und verständlich.

Hinweis: Ein Leerzeichen wird auch als Whitespace bezeichnet.

1.2.6 Groß- und Kleinschreibung

In Sprachen, die zwischen Groß- und Kleinschreibung unterscheiden, würden die folgenden beiden Ausdrücke zwei eigenständige Variablen erzeugen.

Beispiel:

```
<?php
// Variablen
$zahl = 1000;
$Zahl = "tausend";

// Ausgabe 1000
echo $zahl;
?>
```

Der in PHP integrierte Standard erwartet dies auch, die Groß- und Kleinschreibung ist durchgängig zu beachten. Nun kommt es zu einem Problem. PHP wurde kontinuierlich weiter entwickelt. Daher werden auch folgende Schreibweisen zugelassen:

```
<?php
function AkTdAtuM() {
       $zeit = Time();
       $datum = GeTDatE($zeit);
       echo $datum[mday] . ". " . $datum[month] . " " . $datum[year];
}

// Ausführen - 10. December 2003
aktdatum();
?>
```

Besser:

```
<?php
function aktDatum() {
       $zeit = time();
       $datum = getdate($zeit);
       echo $datum[mday] . ". " . $datum[month] . " " . $datum[year];
}

// Ausführen - 10. December 2003
aktDatum();
?>
```

Der Schlüssel zum Erfolg ist es, eine einheitliche Schreibweise einzuhalten. Man sollte im Quelltext einen Stil beibehalten und nicht immer mal groß und mal klein schreiben, da sonst Syntaxfehler sehr leicht entstehen

1.2.7 Geschweifte Klammern

In PHP nehmen die geschweiften Klammern einen besonderen Platz ein. Die PHP-Anweisungen werden, wie im folgenden Skript dargestellt, mit Hilfe geschweifter Klammern ({}) zu Blöcken zusammengefasst:

Beispiel:

```
<?php
$signal = true;

if ($signal == true) {
        //Anweisungsblock
        echo "Signal ist true";
        $signal = false;
}
?>
```

1.2.8 Runde Klammern

Runden Klammern dienen zum Zusammenfassen von bevorzugten Operationen, zum Beispiel um sie anderen Operationen nachzustellen. Außerdem können Sie mit Hilfe von runden Klammern die Reihenfolge der Verarbeitung von PHP-Operationen festlegen.

Beispiel:

```
<?php
// Berechnung wertEins
$wertEins = 10 * 2 + 3;
// Berechnung wertZwei
$wertZwei = 10 * (2 + 3);

// Ausgabe (23)
echo $wertEins;
// Ausgabe (50)
echo $wertZwei;
?>
```

Beim Definieren einer Funktion werden die Parameter in runde Klammern gesetzt.

Beispiel:

```
function meineFunktion ($vorname, $nachname, $anschrift){
        ...
}
```

Bei Aufruf einer Funktion werden alle zu übergebenden Parameter in runde Klammern gesetzt:

```
meineFunktion ("Mike","Müller","Kapweg 10");
```

1.2.9 Schlüsselwörter

Es gibt eine Reihe von reservierten Wörtern in PHP. Das sind Wörter, die im PHP-Code nicht als Bezeichner (als Namen für Variablen, Funktionen etc.) verwendet werden dürfen.

Reservierte PHP-Schlüsselwörter

and	E_PARSE	old_function
$argv	E_ERROR	or
as	E_WARNING	parent
$argc	eval	PHP_OS
break	exit()	$PHP_SELF
case	extends	PHP_VERSION
cfunction	FALSE	print()
class	for	require()
continue	foreach	require_once()
declare	function	return()
default	$HTTP_COOKIE_VARS	static
do	$HTTP_GET_VARS	switch
die()	$HTTP_POST_VARS	stdClass
echo()	$HTTP_POST_FILES	$this
else	$HTTP_ENV_VARS	TRUE
elseif	$HTTP_SERVER_VARS	var
empty()	if	xor
enddeclare	include()	virtual()
endfor	include_once()	while
endforeach	global	__FILE__
endif	list()	__LINE__
endswitch	new	__sleep
endwhile	not	__wakeup
E_ALL	NULL	

Hinweis: Die Tabelle erhebt keinen Anspruch auf Vollständigkeit.

1.2.10 Zuweisungen

Sie haben in den PHP-Beispielen bereits Variablen im Einsatz gesehen. Bei der Zuweisung der Variablenwerte gilt es, Folgendes zu beachten: Wenn man der Variablen `$a` den Wert der Variablen `$b` zuweisen will, muss man dies mit Hilfe des Zuweisungsoperators (Gleichheitszeichen) durchführen. Das bedeutet aber auch, dass man Vergleiche in PHP nicht mit dem einfachen Gleichheitszeichen durchführen kann. Wie man dies erreicht, erfahren Sie noch im Abschnitt »Operatoren«.

Zuweisung (Variable `$a` und `$b`):

```
$a = $b;
```

1.2.11 Echo-Befehl

Der wichtigste Befehl, der Ihnen bei PHP über den Weg laufen wird, ist der `echo`-Befehl. Mit ihm haben Sie die Möglichkeit, Strings auszugeben. Der Text, der ausgegeben werden soll, muss natürlich in Anführungsstrichen stehen, da der Server sonst versucht, ihn als PHP-Befehl zu interpretieren. Dieses Vorgehen wird *Quoten* oder *Quoting* (engl. to quote: zitieren) genannt.

Bei den Anführungszeichen gibt es zwei Arten:

- das einfache Anführungszeichen ' und
- das doppelte Anführungszeichen ".

Es gibt auch einen Unterschied zwischen den beiden:

- Bei den doppelten Anführungsstrichen versucht der Server, den Text zu interpretieren.
- Bei den einfachen hingegen behandelt er ihn nicht speziell, sondern gibt ihn direkt aus.

```
$punkte = 1000;
echo 'Der akt. Punktestand $punkte !\ n';
echo "Der akt. Punktestand $punkte !\ n";
```

Das erste `echo` gibt »*Der akt. Punktestand $punkte !\ n*« aus, das zweite hingegen »*Der akt. Punktestand 1000 !*« mit anschließendem Zeilenumbruch.

```
echo "Mein \" Name \" ist Hase ";
```

Die Ausgabe bei diesem `echo` ist »*Mein »Name« ist Hase*«. Wie man sieht, müssen doppelte Anführungsstriche, die ausgegeben werden sollen, besonders gekennzeichnet werden. Diesen Vorgang nennt man Escapen. Es ist insbesondere für das Ausgeben von HTML-Quelltext in Verbindung mit `echo` und `print` nötig und kann zu Problemen führen, wenn man vergisst, in allen Teilstrings zu »quoten«.

Hinweis: Beim Escapen entkommt das davon betroffene Zeichen der Interpretierung durch den Sprachinterpreter, im vorliegenden Fall PHP.

1.2.12 Print-Befehl

Neben dem `echo`-Befehl gibt es auch den `print`-Befehl. Im Endeffekt leisten beide dasselbe: Sie geben Text aus. `echo` ist ein internes Sprachkonstrukt, wohingegen `print` ein Ausdruck (Expression) ist. `echo` kann mehrere Argumente haben, die nicht in Klammern stehen dürfen. `print` kann nur genau ein Argument haben. Alle folgenden Anweisungen sind zulässig und geben dasselbe aus:

```
$wort1 = "Hallo ";
$wort2 = "Welt !";

echo $wort1 ," ", $wort2;

echo $wort1 ." ". $wortr2;
print ($wort1 ." ". $wort2);

$satz = print ($wort1 ." ". $wort2);
```

1.2.13 Unterschied zwischen echo und print

Im Gegensatz zu `echo` ist `print` eine richtige Funktion. Es gibt Fälle, in denen die Ausgabe mit einer Rückgabe gekoppelt werden muss, beispielsweise beim trinären Bedingungsoperator:

```
<?php
       $wert == 0 ? print "wert ist 0" : print "wert ist nicht 0";
?>
```

Diese Konstruktion wertet eine Bedingung aus und führt entsprechend dem Ergebnis den ersten oder zweiten Teil aus. Es kommt zwar nur auf die Ausgabe an, die Konstruktion erwartet aber Rückgabewerte, der Raum zwischen ? : und ; darf aus syntaktischen Gründen nicht »leer« sein. Hier kann also niemals ein `echo` stehen.

Beispiel:

```
<?php
       $wert == 0 ? echo "wert ist 0" : echo "wert ist nicht 0";
?>
```

Ausgabe:

```
Parse error: parse error, unexpected T_ECHO in C:\php5xampp-
dev\htdocs\php5\ersteschritte.php on line 2
```

Wie Sie sehen, führt der Einsatz von `echo` zu einer Fehlermeldung.

`print` darf nur ein Argument haben. Wenn Sie mehrere Werte haben, nutzen Sie die Schreibweise mit Anführungszeichen oder setzen eine Zeichenkette mit dem Punkt-Operator zusammen.

Hier ist `echo` flexibler, es sind beliebig viele Argumente erlaubt, die jeweils durch ein Komma getrennt werden:

```
<?php
 $vorname = "Matthias";
 echo "Hallo ", $vorname ,"!";
?>
```

Der einzige Unterschied zu der Verknüpfung der Zeichenketten dürfte die Geschwindigkeit sein. In der Praxis ist die Ausgabe jedoch ohnehin der schnellste Teil des Skripts.

Damit bleibt als einziges Argument der optische Eindruck:

```
<?php
 $vorname = "Matthias";
 echo "Hallo " . $vorname . "!";
?>
```

1.2.14 Heredoc

Eine andere Möglichkeit, Strings einzufassen, besteht im Gebrauch der *heredoc*-Syntax (*"<<<"*). Hierfür ist nach <<< ein Bezeichner zu setzen. Nun folgt der eigentliche String und dann derselbe Bezeichner, um den String abzuschließen. Der schließende Bezeichner muss in der ersten Spalte der Zeile stehen.

Achtung: Es ist sehr wichtig zu beachten, dass die Zeile mit dem schließenden Bezeichner keine anderen Zeichen enthält, ausgenommen möglicherweise ein Semikolon (*;*). Das bedeutet, dass der Bezeichner nicht eingerückt werden darf. Es dürfen weiterhin keine Leerzeichen oder Tabulatoren vor oder nach dem Semikolon stehen.

```
echo <<<EINTRAG
Beispiel eines Strings
über mehrere Skript-Zeilen
durch Gebrauch der heredoc-Syntax.
EINTRAG;
```

Heredoc-Text funktioniert wie ein String innerhalb doppelter Anführungszeichen, nur ohne doppelte Anführungszeichen. Anführungszeichen innerhalb von heredoc-Texten müssen also keiner Sonderbehandlung (Escapen) unterzogen werden. Sie können dennoch die oben aufgeführten Escape-Anweisungen verwenden. Variablen werden ausgewertet, aber besondere Aufmerksamkeit muss komplexen Variablen gewidmet werden, genau wie bei Strings.

```
$name = 'Matthias';

echo <<<BUCH
Diese Buch stammt vom <b>$name</b> und stellt Ihnen PHP vor.
BUCH;
```

Hinweis: Die heredoc-Unterstützung wurde in PHP 4 eingeführt.

Die Anwendung ist natürlich nicht auf den `echo`-Befehl beschränkt. Sie können auch einer Variablen einen solchen Block übergeben:

```
<?php
$personen = <<<NAMEN
Matthias Kannengiesser,<br>
Caroline Kannengiesser,<br>
Gülten Kannengiesser
NAMEN;

print $personen;
?>
```

Das sieht unter Umständen viel übersichtlicher und einfacher aus, als wenn die Werte hintereinander stehen. Denken Sie aber daran, dass hier die Zeilenumbrüche erhalten bleiben – nicht in jedem Fall funktioniert das problemlos.

Typische Probleme

Am Anfang passiert es oft, dass der Interpreter nach dem Einbau der Blöcke Fehler meldet. Das folgende Beispiel funktioniert nicht wie erwartet:

```
<?php

$ausgabe = 1;

if ($ausgabe == 1) {
        $personen = <<<NAMEN
        Matthias Kannengiesser,<br>
        Caroline Kannengiesser,<br>
        Gülten Kannengiesser
        NAMEN;

        echo $personen;
}

?>
```

Ausgabe:

```
Parse error: parse error, unexpected $end in C:\php5xampp-
dev\htdocs\php5\ersteschritte.php on line 16
```

Der Fehler liegt nicht direkt im Code, sondern in der Art, wie PHP diesen verarbeitet. Das Ende der heredoc-Sequenz wird nur erkannt, wenn das Schlüsselwort NAMEN am Anfang der Zeile steht. Durch die Einrückung wird es nicht mehr erkannt.

Korrekt:

```
<?php

$ausgabe = 1;

if ($ausgabe == 1) {
        $personen = <<<NAMEN
        Matthias Kannengiesser,<br>
        Caroline Kannengiesser,<br>
        Gülten Kannengiesser
NAMEN;

        echo $personen;
}

?>
```

1.2.15 Kommentare

Jeder kennt die Situation: Man hat eine längere Berechnung durchgeführt, einen Artikel verfasst oder eine Skizze erarbeitet und muss diese Arbeit nun anderen Personen erklären. Leider ist die Berechnung, der Artikel oder die Skizze schon ein paar Tage alt und man erinnert sich nicht mehr an jedes Detail oder jeden logischen Schritt. Wie soll man seine Arbeit auf die Schnelle nachvollziehen?

In wichtigen Fällen hat man deshalb bereits beim Erstellen dafür gesorgt, dass jemand anders (oder man selbst) diese Arbeit auch später noch verstehen kann. Hierzu werden Randnotizen, Fußnoten und erläuternde Diagramme verwendet – zusätzliche Kommentare also, die jedoch nicht Bestandteil des eigentlichen Papiers sind.

Auch unsere Programme werden mit der Zeit immer größer werden. Wir brauchen deshalb eine Möglichkeit, unseren Text mit erläuternden Kommentaren zu versehen. Da sich Textmarker auf dem Monitor schlecht macht, besitzt die Sprache ihre eigenen Möglichkeiten, mit Kommentaren umzugehen.

Jedes Programm, auch das kleinste, sollte sauber kommentiert werden. Eine ordentliche Dokumentation erleichtert die Arbeit während der Entwicklung erheblich. Kommentare sind ein wesentlicher Bestandteil der Dokumentation. Stil und Inhalt sollten dem Programmierer die Übersicht in seinem Code erhalten, aber auch anderen, die mit dem Programm zu tun haben wie zum Beispiel die Teamkollegen bei Projektarbeiten. Denken Sie daran, dass Ihre Überlegungen bei der Umsetzung eines Problems nicht immer ohne weiteres nachvollziehbar sind. Oft gibt es sehr viele Lösungen für eine Idee, die umgesetzt werden soll. Warum Sie eine bestimmte Lösung gewählt haben und wie sie anzuwenden ist, wird in Kommentaren beschrieben.

Kommentare im PHP-Code werden nicht mit übertragen. Der Interpreter ignoriert diese Zeilen und entfernt sie vor der Übertragung. PHP unterstützt Kommentare sowohl nach Art von C/C++ als auch nach Art von Java/JavaScript. Dabei ist zwischen einzeiligen und mehrzeiligen Kommentaren zu unterscheiden. Wir wollen hier nun die unterstützen Kommentare vorstellen.

Im folgenden Kommentar wird festgehalten, wer die Autoren sind.

Einzeiliger Kommentar:

```
// Autor: Matthias Kannengiesser
// Autorin: Caroline Kannengiesser
```

Einzeiliger Kommentar am Zeilenende:

```
$name = "Caroline"; // Dies ist ein Kommentar am Zeilenende
```

Mehrzeiliger Kommentar:

```
/*
-- Anfang Beschreibung --
...
-- Ende Beschreibung --
*/
```

Tipp: Ein Kommentar kann auch dazu verwendet werden, einen Codeabschnitt zeitweilig, ohne direkt ältere Codezeilen zu löschen, zu deaktivieren. Dieses Vorgehen bezeichnet man in der Programmierung als Auskommentieren. Ein Vorteil beim Auskommentieren von Code liegt darin, diesen zu einem späteren Zeitpunkt, ohne größeren Aufwand, wieder herzustellen bzw. aktivieren zu können, indem die Kommentarzeichen entfernt werden.

Für den letzten Fall gibt es noch eine weitere Variante. Mit dem Zeichen # können ebenfalls Kommentare gekennzeichnet werden:

```
$vorname = "Matthias"; # Name des Autors
```

Hier ein Beispiel, in dem alle drei Kommentararten vorkommen:

```
<?php
/*

Hier ein mehrzeiliger Kommentar
im PHP-Code

*/

$vorname = "Matthias"; # Vornname des Autors
$nachname = "Kannengiesser"; // Nachname des Autors

echo $vorname /* Ausgabe */
?>
```

Was letztlich zum Einsatz kommt, bleibt ganz Ihnen überlassen.

1.3 Datentypen

Daten sind nicht gleich Daten. Nehmen Sie zum Beispiel die Daten 1000 und »Gülten«. Sicherlich werden Sie erkennen, dass Sie es hier mit unterschiedlichen Typen von Daten zu tun haben:

10 ist eine Zahl.

»Gülten« ist eine Folge von Zeichen.

Auch PHP trifft diese Unterscheidung und geht dabei sogar noch einen Schritt weiter. PHP unterstützt acht so genannte »primitive« Typen.

Vier skalare Typen:

- Boolean
- Integer
- Fließkommazahl (float)
- String / Zeichenkette

Zwei zusammengesetzte Typen:

- Array
- Object

Und zuletzt zwei spezielle Typen:

- Resource
- NULL

Ein Datentyp beschreibt die Art der Informationen, die eine Variable oder ein PHP-Element enthalten kann.

Der Typ einer Variablen wird normalerweise nicht vom Programmierer bestimmt. Vielmehr wird zur Laufzeit von PHP entschieden, welchen Datentyp eine Variable erhält, abhängig vom Kontext, in dem die Variable verwendet wird.

1.3.1 Strings/Zeichenketten

Ein String oder auch Zeichenkette genannt ist eine Folge von Buchstaben, Ziffern und Sonderzeichen. Ein String wird von Anführungszeichen umschlossen, entweder von einfachen (Apostrophen) oder doppelten. Dabei ist zu beachten, dass unbedingt gerade Anführungszeichen [Umschalt]+[2] und Apostrophe [Umschalt]+[#] verwendet werden. Im Gegensatz zu anderen Programmiersprachen ist es egal, ob einfache oder doppelte Anführungszeichen verwendet werden, Hauptsache, die Zeichenkette wird mit derselben Art von Anführungszeichen beendet und eingeleitet.

```
$meineMutter = "Gülten";
$meineSchwester = 'Caroline';
```

Die verschiedenen Anführungszeichen haben unter anderem den folgenden Sinn: Wenn Sie beispielsweise einen Apostroph in einer Zeichenkette verwenden wollen, können Sie diese Zeichenkette schlecht mit Apostrophen eingrenzen, da der PHP-Interpreter dann nicht weiß, wo die Zeichenkette aufhört. In diesem Fall müssen Sie die andere Sorte von Anführungszeichen verwenden.

Beispiel:

```
// Fehlerhaft
$spruch = 'Ich bin's!';
// Korrekt
$spruch = "Ich bin's!";
```

Wenn man aber in die Verlegenheit kommt, beide Arten von Anführungszeichen in einer Zeichenkette verwenden zu müssen, kommt man in Schwierigkeiten. Hier hilft der Backslash (\) weiter. Das dem Backslash folgende Zeichen wird entwertet, d.h., es nimmt in der Zeichenkette keine besondere Bedeutung ein. Beim Anführungszeichen oder Apostroph bedeutet das, dass die Zeichenkette hiermit nicht beendet wird.

Beispiel:

```
// Backslash
$spruch = 'Ich bin\'s!';
```

Wenn man nun den Backslash selbst in der Zeichenkette verwenden will, muss man auch ihn entwerten.

```
// Verzeichnis
$dateiPfad = "C:\\PROGRAMME";
```

Hinweis: Sollten Sie vorhaben, andere Zeichen zu escapen, wird der Backslash ebenfalls ausgegeben! Daher besteht gewöhnlich keine Notwendigkeit, den Backslash selbst zu escapen.

Die Kombination eines Zeichens mit einem vorangestellten Backslash wird übrigens als Escape-Sequenz bezeichnet. Neben den Anführungszeichen können noch eine Reihe weiterer Zeichen in PHP nur durch eine solche Escape-Sequenz dargestellt werden, die wir Ihnen weiter unten in diesem Abschnitt vorstellen.

Syntax

Ein String kann auf dreierlei Art und Weise geschrieben werden:

- einfache Anführungszeichen (single quoted)
- doppelte Anführungszeichen (double quoted)
- Heredoc-Syntax

Einfache Anführungszeichen (single quoted)

Der leichteste Weg, einen einfachen String zu schreiben, ist das Einschließen in einfache Anführungszeichen (').

```
<?php
// Ausgabe - PHP 5 lässt es krachen
echo 'PHP 5 lässt es krachen';

// Ausgabe - Herzlich Willkommen, Wir sind Ihre...
echo 'Herzlich Willkommen,
         Wir sind Ihre...';
?>
```

Anders als bei den zwei anderen Schreibweisen werden Variablen innerhalb von single-quoted Strings nicht ausgewertet.

```
<?php
$person = "Caroline";

// Ausgabe - Guten Morgen, $person
echo 'Guten Morgen, $person';
?>
```

Doppelte Anführungszeichen (double quoted)

Wenn ein String in doppelte Anführungszeichen (") gesetzt wird, versteht PHP mehr Escape-Folgen für spezielle Zeichen:

Zeichenfolge	*Bedeutung*
\n	Zeilenvorschub (LF oder 0x0A als ASCII-Code).
\r	Wagenrücklauf (CR oder 0x0D als ASCII-Code).
\t	Tabulator (HT oder 0x09 als ASCII-Code).
\\	Backslash bzw. Rückstrich.
\$	Dollar-Symbol.
\ '	Einfaches Anführungszeichen.
\"	Doppeltes Anführungszeichen.
\[0-7]{1,3}	Die Zeichenfolge, die dem regulären Ausdruck entspricht, ist ein Zeichen in Oktal-Schreibweise.
\x[0-9A-Fa-f]{1,2}	Die Zeichenfolge, die dem regulären Ausdruck entspricht, ist ein Zeichen in Hexadezimal-Schreibweise.

Sollten Sie versuchen, sonstige Zeichen zu escapen, wird der Backslash ebenfalls ausgegeben.

Der wohl wichtigste Vorteil von double-quoted Strings ist die Tatsache, dass Variablen ausgewertet werden.

```
<?php
$person = "Caroline";

// Ausgabe - Guten Morgen, Caroline
echo "Guten Morgen, $person";
?>
```

Heredoc

Der Einsatz von Heredoc wurde bereits im Abschnitt »Einführung in PHP« beschrieben, daher hier lediglich einige weitere Besonderheiten.

Heredoc-Text funktioniert wie ein String innerhalb doppelter Anführungszeichen, nur ohne doppelte Anführungszeichen. Anführungszeichen innerhalb von heredoc-Texten müssen also keiner Sonderbehandlung (Escapen) unterzogen werden, aber Sie können dennoch die oben aufgeführten Escape-Anweisungen verwenden. Variablen werden ausgewertet.

```
<?php
echo <<<ANREDE
Herzlich Willkommen,
Meine Damen und Herren...
ANREDE;
?>
```

Variablen-Analyse (parsing)

Wird ein String in doppelten Anführungszeichen oder mit `heredoc` angegeben, werden enthaltene Variablen ausgewertet (geparst).

Es gibt zwei Syntaxtypen, eine einfache und eine komplexe.

- Die einfache Syntax ist die geläufigste und bequemste. Sie bietet die Möglichkeit, eine Variable, einen Array-Wert oder eine Objekt-Eigenschaft auszuwerten (parsen).
- Die komplexe Syntax wurde in PHP 4 eingeführt und ist an den geschweiften Klammern {}erkennbar, die den Ausdruck umschließen.

Einfache Syntax

Sobald ein Dollarzeichen ($) auftaucht, wird der Parser versuchen, einen gültigen Variablennamen zu bilden. Schließen Sie Ihren Varaiblennamen in geschweifte Klammern ein, wenn Sie ausdrücklich das Ende des Namens angeben wollen.

```
<?php
$marke = 'Audi';

/*
 Ausgabe - Auid's sind goldig
 Da ' kein gültiges Zeichen für einen
 Variablennamen darstellt.
*/
echo "$marke's sind goldig";

/*
 Ausgabe - Sie haben zahlreiche gefahren
 Da s ein gültiges Zeichen für einen
 Variablennamene darstellt, wird der
 Interpreter nach einer Variablen mit
 dem Namen $markes suchen.
*/
echo "Sie haben zahlreiche $markes gefahren";

//Ausgabe - Sie hanen zahlreiche Audis gefahren
echo "Sie haben zahlreiche ${marke}s gefahren";
?>
```

Auf ähnliche Weise können Sie erreichen, dass ein Array-Index oder eine Objekt-Eigenschaft ausgewertet wird. Bei Array-Indizes markiert die schließende eckige Klammer (]) das Ende des Index. Für Objekt-Eigenschaften gelten die gleichen Regeln wie bei einfachen Variablen, obwohl es bei Objekt-Eigenschaften keinen Trick gibt, wie dies bei Variablen der Fall ist.

```
<?php
$autos = array( 'Viper' => 'gelb' , 'Ferrari' => 'rot' );

/*
 Ausgabe - Ein Ferarri ist rot
 Achtung: außerhalb von String-Anführungszeichen funktioniert das anders.
```

```
*/
echo "Ein Ferarri ist $autos[Ferrari].";
?>
```

```
<?php
// Klasse
class Fahrzeug
{
    var $plaetze;

    function Fahrzeug()
    {
        $this->plaetze = 4;
    }
}

// Objekt
$meinauto = new Fahrzeug;

// Ausgabe - Dieses Auto hat Platz für 4 Personen.
echo "Dieses Auto hat Platz für $meinauto->plaetze Personen.";

// Ausgabe - Dieses Auto hat Platz für Personen.
// Funktioniert nicht. Für eine Lösung siehe die komplexe Syntax.
echo "Dieses Auto hat Platz für $meinauto->plaetze00 Personen.";
?>
```

Für komplexere Strukturen sollten Sie die komplexe Syntax verwenden.

Komplexe (geschweifte) Syntax

Diese Syntax wird nicht komplex genannt, weil etwa die Syntax komplex ist, sondern weil Sie auf diesem Weg komplexe Ausdrücke einbeziehen können.

Tatsächlich können Sie jeden beliebigen Wert einbeziehen, der einen gültigen Namensbereich als String besitzt. Schreiben Sie den Ausdruck einfach auf die gleiche Art und Weise wie außerhalb des Strings, und umschließen Sie diesen mit { und }. Da Sie '{' nicht escapen können, wird diese Syntax nur erkannt, wenn auf { unmittelbar $ folgt. Benutzen Sie »{\$« oder »\{$«, um ein wörtliches »{$« zu erhalten. Hier ein Beispiel:

```
<?php
// Klasse
class Fahrzeug
{
    var $plaetze;

    function Fahrzeug()
    {
        $this->plaetze = 4;
    }
}

// Objekt
```

```
$meinauto = new Fahrzeug;

// Ausgabe - Dieses Auto hat Platz für 400 Personen.
echo "Dieses Auto hat Platz für {$meinauto->plaetze}00 Personen.";
?>
```

Umwandlung von Zeichenketten

Sobald ein String als numerischer Wert ausgewertet wird, wird der resultierende Wert und Typ wie folgt festgelegt. Der String wird als `float` ausgewertet, wenn er eines der Zeichen '.', 'e' oder 'E' enthält. Ansonsten wird er als Integer-Wert interpretiert.

Der Wert wird durch den Anfangsteil des Strings bestimmt. Sofern der String mit gültigen numerischen Daten beginnt, werden diese als Wert benutzt. Andernfalls wird der Wert 0 (Null) sein. Gültige numerische Daten sind ein optionales Vorzeichen, gefolgt von einer oder mehreren Zahlen (optional mit einem Dezimalpunkt). Wahlweise kann auch ein Exponent angegeben werden. Der Exponent besteht aus einem 'e' oder 'E', gefolgt von einer oder mehreren Zahlen.

```
<?php
$wert = 1 + "10.5";
// Ausgabe - $wert ist float (11.5)
echo $wert;

$wert2 = 1 + "Matze3";
// Ausgabe - $wert2 ist integer (1)
echo $wert2;

$wert3 = "10 Autos" + 1;
// Ausgabe - $wert3 ist integer (11)
echo $wert3;
?>
```

1.3.2 Zahlen

Zahlen sind der grundlegendste Datentyp überhaupt und benötigen wenig Erläuterung. Es kann sich dabei um ganze Zahlen oder Fließkommazahlen handeln. Ganze Zahlen (Integer) können dezimal, oktal oder hexadezimal geschrieben werden.

Der Datentyp für die Zahlen umfasst:

- ganzzahlige Zahlenwerte wie 10, 50, 1000000 oder –1000, die auch als Integer-Zahlen bezeichnet werden;
- Zahlen mit Nachkommastellen wie 9.95 oder 3.1415, die auch als Gleitkommazahlen (Float-Zahlen) bezeichnet werden.

Achtung: Sie sollten dabei beachten, dass der Punkt hier zur Abtrennung der Nachkommastellen und nicht wie im Deutschen üblich zur Kennzeichnung der Tausenderstellen dient.

Wie sieht es nun mit der Schreibweise von ganzzahligen Werten aus?

Beispiel:

```
$zahleins = 9000;
```

Dieser Zahlenwert kann auch als Hexadezimalzahl angegeben werden. Um dem PHP-Interpreter anzuzeigen, dass die folgende Zahl eine Hexadezimalzahl ist, stellen Sie dem Zahlenwert das Präfix 0x voran.

Beispiel:

```
$zahlEins = 0x2328;  // entspricht dezimal 9000
```

Nun zu den Fließkommazahlen:

```
$zahlzwei = 999.99;
```

Wie Sie sehen, werden Fließkommazahlen mit einem Punkt zwischen den Vorkomma- und Nachkommastellen geschrieben. Alternativ können Sie Fließkommazahlen auch in der Exponentialschreibweise angeben.

```
$zahldrei = 999.99e2; // entspricht 99999
```

Hinweis: Der Buchstabe e wird in Fließkommazahlen zur Kennzeichnung eines nachfolgenden Exponenten zur Basis 10 verwendet. 999.99e2 bedeutet also $999.99 * 10^2$, nicht zu verwechseln mit der Eulerschen Zahl.

PHP-Programme bearbeiten Zahlen, indem sie die arithmetischen Operatoren benutzen, die die Sprache zur Verfügung stellt. Dazu gehören:

Addition (+)

Substraktion (-)

Multiplikation (*)

Division (/)

Modulo (%)

Die vollständigen Einzelheiten bezüglich dieser und anderer arithmetischer Operatoren finden sich im Abschnitt »Operatoren«.

Zusätzlich zu diesen grundlegenden mathematischen Operationen unterstützt PHP komplizierte mathematische Operationen durch eine große Anzahl an mathematischen Funktionen, die zu den Kernbestandteilen der Sprache gehören.

Beispiel:

```
// Sinus von x berechnen
$sinusx = sin($x);
```

Die Anweisung ermöglicht es Ihnen, den Sinus eines Zahlenwerts x zu berechnen. Im Abschnitt der praktischen Anwendungsbeispiele werden wir einige nützliche Formeln aufzeigen.

Integer-Typen

In der Mathematik würde der Integer-Typ dem Wertebereich aus der Menge Z = {..., -2, -1, 0, 1, 2, ...} entsprechen. Wie bereits erwähnt, können Ganzzahlen in dezimaler (10-basierter), hexadezimaler (16-basierter) oder oktaler (8-basierter) Schreibweise angegeben werden, wahlweise mit einem vorangestellten -/+ Zeichen.

Schreibweisen:

- Für Oktalzahlen wird eine vorangestellte 0 benötigt.
- Für Hexadezimalzahlen wird ein vorangestelltes 0x benötigt.

Beispiel:

```
<?php
// Dezimalzahl
$zahl = 1234;
echo $zahl;

// Negative Zahl
$zahl = -123;
echo $zahl;

// Oktalzahl (87)
$zahl = 0127;
echo $zahl;

// Hexadezimalzahl (255)
$zahl = 0xFF;
echo $zahl;
?>
```

Die Größe eines Integer-Werts ist von der jeweiligen Plattform abhängig, ein Maximalwert von ungefähr zwei Milliarden ist jedoch üblich (32 Bit).

Integer-Überlauf

Sollten Sie eine Zahl jenseits der Grenzen des Typs `integer` angeben, wird diese als Typ `float` interpretiert. Wenn Sie eine Operation ausführen, deren Ergebnis eine Zahl jenseits der Grenzen des Typs `integer` ist, wird ebenso eine Zahl vom Typ `float` zurückgegeben.

Beispiel:

```
<?php
// Ausgabe: int(2147483647)
$zahl =  2147483647;
var_dump($zahl);

// Ausgabe: float(2147483648)
$zahl =  2147483648;
var_dump($zahl);
?>
```

Dies gilt auch für Integer in hexadezimaler Schreibweise.

Beispiel:

```
<?php
// Ausgabe: int(2147483647)
var_dump(0x7fffffff);

// Ausgabe: float(2147483648)
var_dump(0x80000000);
?>
```

Und für Integer, welche aus einer Rechenoperation resultieren.

Beispiel:

```
<?php
// Ausgabe: float(1.0E+11)
$zahl =  100000 * 1000000;
var_dump($zahl);
?>
```

In PHP gibt es keinen Operator für Integer-Divisionen. 1/2 ergibt 0.5.

Beispiel:

```
<?php
// Ausgabe: float(0.5)
var_dump(1/2);
?>
```

Umwandlung in integer

Um einen Wert ausdrücklich in einen Integer zu konvertieren, benutzen Sie entweder die Umwandlung mittels (`int`) oder (`integer`). In den allermeisten Fällen ist es jedoch nicht notwendig, die Umwandlung selbst vorzunehmen. Ein Wert wird automatisch konvertiert, falls ein Operator, eine Funktion oder eine Kontrollstruktur ein `integer`-Argument erfordert.

Beispiel:

```
<?php
// Ausgabe (10)
echo (int) 10.99;
// Ausgabe (10)
echo (integer) 10.99;
?>
```

Umwandlung von Booleans in integer

FALSE ergibt 0 (Null), und TRUE ergibt 1 (Eins).

Beispiel:

```
<?php
// Boolean
echo (int) true;  // 1
echo (int) false; // 0
?>
```

Umwandlung von Fließkomma-Zahlen in integer

Bei der Umwandlung von `float` nach `integer` wird die Zahl in Richtung Null gerundet.

Beispiel:

```
<?php
// Ausgabe (99999)
echo (int) 9999.4567;
?>
```

Wenn der `float` jenseits der Grenzen von `integer` liegt (üblicherweise +/- 2.15e+9 = 2^31), ist das Ergebnis nicht definiert, da `float` nicht genug Präzision besitzt, um ein genaues `integer`-Ergebnis zu liefern. Keine Warnung oder Fehlermeldung wird in diesem Fall ausgegeben.

Sie sollten nie einen Teilausdruck nach `integer` umwandeln, da dies in einigen Fällen zu unerwarteten Ergebnissen führen kann.

Beispiel:

```
<?php
// Ausgabe (8)
echo ( (0.7+0.1) * 10 );
// Ausgabe (7)
echo (int) ( (0.7+0.1) * 10 );
?>
```

Umwandlung von Strings in integer

Natürlich lassen sich auch Zeichenketten in integer umwandeln.

```
<?php
// Ausgabe (10)
echo 2 * "5 Äpfel";

// Ausgabe (6)
echo  "5 Autos " + 1;

// Ausgabe (1)
echo 1 + "C-64";
?>
```

Float-Typen

Fließkommazahlenwerte, auch als Floats, Doubles oder reelle Zahlen bezeichnet, können durch eine der folgenden Anweisungen zugewiesen werden:

```
<?php
// Ausgabe (1.234)
$wert = 1.234;
echo $wert;

// Ausgabe (1200)
$wert = 1.2e3;
echo $wert;

// Ausgabe (7E-10)
$wert = 7E-10;
echo $wert;
?>
```

Die Größe einer Fließkommazahl ist plattformabhängig, dennoch stellt ein Maximum von ~1.8e308 mit einer Genauigkeit von 14 Nachkommastellen einen üblichen Wert dar (das entspricht 64 Bit im IEEE-Format).

Fließkomma-Präzision

Es ist ziemlich normal, dass einfache Dezimalzahlen wie 0.1 oder 0.7 nicht in ihre internen binären Entsprechungen konvertiert werden können, ohne einen kleinen Teil ihrer Genauigkeit zu verlieren. Das kann zu verwirrenden Ergebnissen führen.

```
<?php
// Ausgabe (7)
echo floor((0.1 + 0.7) * 10)
?>
```

Sie haben sicher 8 erwartet. Dieses Ergebnis stützt sich auf die Tatsache, dass es unmöglich ist, einige Dezimalzahlen durch eine endliche Anzahl an Nachkommastellen darzustellen. Dem Wert 1/3 entspricht z.B. der interne Wert von 0.3333333.

Daher sollten Sie nie den Ergebnissen von Fließkomma-Operationen bis auf die letzte Nachkommastelle trauen, sondern sie auf Gleichheit prüfen.

Tipp: Benötigen Sie eine größere Genauigkeit, sollten Sie die mathematischen Funktionen beliebiger Genauigkeit oder die Gmp-Funktionen verwenden.

1.3.3 Boolean/Boolesche Werte

Die Datentypen für Zahlen und Strings können beliebig viele verschiedene Werte annehmen. Der Datentyp `boolean` kennt hingegen nur zwei mögliche Werte. Die zwei zulässigen booleschen Werte sind `true` (wahr) und `false` (falsch). Ein boolescher Wert stellt einen Wahrheitswert dar, der besagt, ob ein Sachverhalt wahr ist oder nicht.

Hinweis: Die beiden Schlüsselwörter TRUE oder FALSE unterscheiden nicht zwischen Groß- und Kleinschreibung.

Tipp: Um boolesche Werte besser zu verstehen, sollte man sich das einfache Schema eines Lichtschalters vorstellen. Ist das Licht an, steht der Lichtschalter auf »Ein«, dies entspricht dem booleschen Wert `true`. Ist das Licht aus, steht der Lichtschalter auf »Aus«, dies entspricht dem booleschen Wert `false`. Natürlich kann dieses Schema nur angewendet werden, wenn die Birne in der Lampe in Ordnung ist und der Stecker steckt!

Beispiel:

```
// Licht ist eingeschaltet
$licht = true;
// Licht ist ausgeschaltet
$licht = false;
```

Boolesche Werte sind im Allgemeinen das Ergebnis von Vergleichen, die in einem Programm vorgenommen werden. Wie sieht ein solcher Vergleich aus?

```
$name == "Matthias";
```

Hier sehen Sie einen Teil eines Vergleichs. Dabei wird überprüft, ob der Wert der Variablen `name` der Zeichenkette »Matthias« entspricht. Sollte dies der Fall sein, ist das Ergebnis des Vergleichs der boolesche Wert `true`. Wenn der Wert der Variablen `name` nicht der Zeichenkette entsprechen sollte, dann ist das Ergebnis des Vergleichs `false`. Boolesche Werte werden in PHP gewöhnlich durch Vergleiche erzeugt und zur Ablaufsteuerung genutzt.

In einer If-Else-Konstruktion wird eine Aktion ausgeführt, wenn ein boolescher Wert `true` ist, aber eine andere, wenn dieser Wert `false` ist. Häufig wird ein Vergleich, der einen booleschen Wert erzeugt, unmittelbar mit einer Anweisung kombiniert, die diesen Wert benutzt.

Beispiel:

```
<?php
if ($name == "Matthias") {
        $spruch = "Hallo " + $name;
        echo $spruch;
} else {
        $spruch = "Sie kenne ich nicht!";
        echo $spruch;
}
?>
```

Dieses Beispiel prüft, ob der Wert in der Variablen `$name` der Zeichenkette »Matthias« entspricht. Wenn ja, dann wird in die Variable `$spruch` die Zeichenkette »Hallo Matthias« eingesetzt, sonst wird in die Variable `$spruch` die Zeichenkette »Sie kenne ich nicht!« eingesetzt.

Eines sollte hier zu den booleschen Werten noch erwähnt werden: PHP ist in der Lage, die Zahlenwerte 1 und 0 als boolesche Werte `true` und `false` zu interpretieren.

Beispiel:

```
$signaleins = true;
$signalzwei = 1;
```

Beide Anweisungen enthalten unterschiedliche Datentypen, jedoch erst aus dem Kontext heraus wird ersichtlich, ob es sich bei der Variablen `$signalzwei` um einen Zahlenwert oder einen booleschen Wert handelt.

Beispiel:

```
<?php
// Variable
$signalzwei = 1;

// Zahl
$summe = $signalzwei + 5;

// Boolscher Wert
if ($signalzwei == true) {
        $zustand = "Signal ist Ein";
        echo $zustand;
}
?>
```

Achtung: C/C++-Programmierer sollten beachten, dass PHP einen eigenen Datentyp `Boolean` hat. Dies steht im Gegensatz zu C/C++, die einfache ganzzahlige Werte benutzen, um boolesche Werte nachzuahmen.

Umwandlung nach boolean

Um einen Wert ausdrücklich nach boolean zu konvertieren, benutzen Sie entweder die Umwandlung mittels (`bool`) oder (`boolean`).

Beispiel:

```
<?php
// Ausgabe (1)
echo (bool) ((100));
// Ausgabe (1)
echo (boolean) ((100));
?>
```

In den allermeisten Fällen ist es jedoch nicht notwendig, die Umwandlung selbst vorzunehmen. Ein Wert wird automatisch konvertiert, falls ein Operator, eine Funktion oder eine Kontrollstruktur ein boolean Argument erfordert.

Bei der Umwandlung nach boolean werden folgende Werte als FALSE angesehen:

- das boolean FALSE
- die Integer 0 (Null)
- die Fließkommazahl 0.0 (Null)

- die leere Zeichenkette und die Zeichenkette »0«
- ein Array ohne Elemente
- ein Objekt ohne Elemente
- der spezielle Type NULL (einschließlich nicht definierter Variablen)

Jeder andere Wert wird als TRUE angesehen, einschließlich jeder Ressource.

Achtung: -1 wird als TRUE angesehen, wie jede andere Zahl ungleich Null. Ob es sich dabei um eine negative oder positive Zahl handelt ist nicht relevant.

1.3.4 Objekte

Objekte gehören dem PHP-Datentyp `Object` an. Ein Objekt ist eine Sammlung benannter Daten. Die Namen dieser Datenelemente werden als Eigenschaften, Attribute oder Propertys des Objekts bezeichnet. Einige sprechen auch von den Feldern des Objekts. Diese Benennung kann jedoch zu Verwirrungen führen, da so genannte Felder (Arrays) auch als eigenständiger Datentyp existieren. Sie sollten sich daher auf eine der ersten drei Benennungen stützen. Um die Eigenschaft eines Objekts anzusprechen, sprechen wir zunächst das Objekt an, setzen dahinter ein -> und fügen anschließend den Namen der Eigenschaft an.

Beispiel:

```
$meinRechner->hersteller
$meinRechner->cpu
```

Hier haben wir ein Objekt namens `$meinRechner` mit den Eigenschaften `hersteller` und `cpu`. Die Eigenschaften von Objekten verhalten sich dabei in PHP wie Variablen. Sie können alle Datentypen enthalten, einschließlich Arrays, Funktionen und anderer Objekte. Wie bereits erwähnt, wird eine Funktion, die als Eigenschaft eines Objekts gespeichert ist, oft auch als Methode bezeichnet. Um eine Methode eines Objekts aufzurufen, benutzen Sie wiederum die Schreibweise mit dem ->, um den Funktions-Datenwert in dem Objekt ansprechen zu können.

Beispiel:

```
$meinRechner->starten();
```

Wir wollen Ihnen noch ein lauffähiges Beispiel für die aufgeführten Eigenschaften und die Funktion mit auf den Weg geben:

```
<?php
/*
 Klasse (class)
 Definiert Eigenschaften, Methoden und Funktionen
 einer Gruppe von Objekten
*/
class Rechner {

       var $cpu;
```

```
        var $hersteller;

        function Rechner($taktrate,$unternehmen)
    {
         $this->cpu = $taktrate;
         $this->hersteller = $unternehmen;
    }
    function starten()
    {
                echo "Rechner gestartet!";
    }
}

// Objekterzeugen
$meinRechner = new Rechner(2000,"Intel");

// Ausgabe - Object id #1
echo $meinRechner;
// Ausgabe - Intel
echo $meinRechner->hersteller;
// Ausgabe - Rechner gestartet!
echo $meinRechner->starten();
?>
```

1.3.5 Arrays

Ein Array in PHP ist im eigentlichen Sinne eine geordnete Abbildung. Eine Abbildung ist ein Typ, der Werte auf Schlüssel abbildet. Dieser Typ ist auf mehrere Arten optimiert, so dass Sie ihn auf verschiedene Weise benutzen können:

- als reales Array
- als Liste (Vektor)
- als Hash-Tabelle
- als Verzeichnis
- als Sammlung
- als Stapel (Stack)
- als Warteschlange (Queue)

und vieles mehr.

Da Sie ein weiteres PHP-Array als Wert benutzen können, ist es recht einfach, Baumstrukturen zu simulieren und Verschachtelungen vorzunehmen.

Angabe mit array()

Ein Array kann mit Hilfe des Sprachkonstrukts `array()` erzeugt werden. Es benötigt eine bestimmte Anzahl von durch Komma getrennten `Schlüssel => Wert`-Paaren.

Ein Schlüssel ist entweder eine Zahl vom Typ `integer` oder ein String. Wenn ein Schlüssel die Standarddarstellung einer Integer-Zahl ist, wird er als solche interpretiert wird:

- »8« wird als 8 interpretiert.
- »08« wird als »08« interpretiert.

Wert

Der Wert eines Arrays-Eintrags kann ein beliebiger Datentyp sein.

Schlüssel

Falls Sie einen Schlüssel weglassen, wird das Maximum des Integer-Indizes genommen und der neue Schlüssel wird das Maximum + 1 sein. Das gilt auch für negative Indizes, da ein Integer negativ sein kann. Ist zum Beispiel der höchste Index -6, wird der neue Schlüssel den Wert -5 haben. Falls es bis dahin keine Integer-Indizes gibt, wird der Schlüssel zu 0 (Null). Falls Sie einen Schlüssel angeben, dem schon ein Wert zugeordnet wurde, wird dieser Wert überschrieben.

Wenn Sie `true` als Schlüssel benutzen, wird dies als Schlüssel vom Typ `integer 1` ausgewertet. Benutzen Sie `false` als Schlüssel, wird dies als Schlüssel vom Typ `integer 0` ausgewertet. Die Benutzung von `NULL` als Schlüssel führt dazu, dass der Schlüssel als leerer String gewertet wird. Verwenden Sie einen leeren String als Schlüssel, wird ein Schlüssel mit einem leeren String und seinem Wert erzeugt oder überschrieben. Das entspricht nicht der Verwendung von leeren Klammern.

Sie können keine Arrays oder Objekte als Schlüssel benutzen. Der Versuch wird mit einer Warnung enden: `Illegal offset type`.

```
// Schlüssel ist entweder ein string oder integer
// Wert kann irgendetwas sein.
array( [Schlüssel =>] Wert, ...)
```

Hier einige Beispiele:

```
<pre>
<?php
$arrays = array (
    "Fruechte" => array ("a"=>"Kirsche", "b"=>"Birne"),
    "Zahlen"  => array (1, 2, 3, 4, 5, 6),
    "Autos"  => array ("Audi", 5 => "Mercedes", "BMW")
);
print_r($arrays);
?>
</pre>
```

Ausgabe:

```
Array
(
    [Fruechte] => Array
```

```
        (
            [a] => Kirsche
            [b] => Birne
        )

    [Zahlen] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 3
            [3] => 4
            [4] => 5
            [5] => 6
        )

    [Autos] => Array
        (
            [0] => Audi
            [5] => Mercedes
            [6] => BMW
        )

)
```

Automatischer Index mit array():

```
<pre>
<?php
$meinarray = array( 10,
                                    20,
                                    30,
                                    40,
                                    50,
                                    60,
                                    70,
                                    4=>1,
                                    5=>1,
                                    6=>13
                                    );
print_r($meinarray);
?>
</pre>
```

Ausgabe:

```
Array
(
    [0] => 10
    [1] => 20
    [2] => 30
    [3] => 40
    [4] => 1
    [5] => 1
    [6] => 13
)
```

Auf 1 basierter Index mit array():

```
<pre>
<?php
$monate  = array(1 => 'Januar', 'Februar', 'März');
print_r($monate);
?>
</pre>
```

Ausgabe:

```
Array
(
    [1] => Januar
    [2] => Februar
    [3] => März
)
```

Erzeugen/Verändern mit der eckigen Klammer-Syntax

Sie können ein bestehendes Array durch explizites Zuweisen von Werten verändern. Weisen Sie dem Array Werte zu, indem Sie den Schlüssel in eckigen Klammern angeben. Sie können den Schlüssel auch weglassen. In diesem Fall schreiben Sie einfach ein leeres Klammerpaar ("[]") hinter den Variablennamen.

```
// Schlüssel ist entweder ein string oder ein nicht-negativer integer
// Wert kann irgendetwas sein.
$arr[Schlüssel] = Wert;
$arr[] = Wert;
```

Beispiel:

```
<pre>
<?php
$monate  = array(1 => 'Januar', 'Februar', 'März');
$monate[4] = "April";
$monate[] = "Mai";
print_r($monate);
?>
</pre>
```

Ausgabe:

```
Array
(
    [1] => Januar
    [2] => Februar
    [3] => März
    [4] => April
    [5] => Mai
)
```

Falls das Array bis dahin nicht existiert, wird es erzeugt. Das ist also eine alternative Syntax, um ein Array zu erzeugen. Um einen bestimmten Wert zu ändern, weisen Sie ihm einfach einen neuen Wert zu.

Beispiel:

```
<pre>
<?php
$personen[0] = "VW";
$personen[1] = "BMW";
$personen[2] = "OPEL";
print_r($personen);
?>
</pre>
```

Ausgabe:

```
Array
(
    [0] => VW
    [1] => BMW
    [2] => OPEL
)
```

Sollten Sie ein Schlüssel/Wert-Paar entfernen wollen, benutzen Sie `unset()`.

Beispiel:

```
<pre>
<?php
$personen[0] = "VW";
$personen[1] = "BMW";
$personen[2] = "OPEL";

// Löschen
unset ($personen[1]);
print_r($personen);
?>
</pre>
```

Ausgabe:

```
Array
(
    [0] => VW
    [2] => OPEL
)
```

Nützliche Funktionen

Es existieren eine Vielzahl von nützlichen Funktionen, um mit Arrays zu arbeiten. Einige davon werden wir im Abschnitt »Arrays« gesondert behandeln.

Beispiel:

```
<?php
// Array
$daten[0] = 100;
$daten[1] = 300;
$daten[2] = 500;
```

```
$anzahl = count($daten);
// Ausgabe (3)
echo $anzahl;
?>
```

Schreibweisen und ihre Besonderheiten

Warum ist `$daten[eintrag]` falsch? – Sie sollten immer Anführungszeichen für einen assoziativen Index eines Arrays benutzen. Zum Beispiel sollten Sie `$daten['eintrag']` und nicht `$daten[eintrag]` benutzen. Aber warum ist `$daten[eintrag]` falsch? Sicher ist Ihnen folgende Syntax in einigen PHP-Skripts bereits begegnet:

```
<?php
$daten[eintrag] = "Matze";
// Ausgabe - Matze
echo $daten[eintrag];
?>
```

Diese Schreibweise funktioniert zwar, ist aber falsch. Der Grund ist, dass dieser Code eine undefinierte Konstante `eintrag` enthält anstatt eines Strings 'eintrag'. Beachten Sie die Anführungszeichen. PHP könnte in Zukunft Konstanten definieren, die unglücklicherweise für ihren Code den gleichen Namen verwenden. Es funktioniert, weil die undefinierte Konstante in einen String mit gleichem Namen umgewandelt wird. PHP nimmt an, dass Sie `eintrag` wörtlich gemeint haben, wie den String `'eintrag'`, aber vergessen haben, die Anführungszeichen zu setzen.

Daher ist die folgende Schreibweise zu bevorzugen:

```
<?php
$daten['eintrag'] = "Matze";
// Ausgabe - Matze
echo $daten['eintrag'];
?>
```

Hinweis: Noch mehr rund um das Thema Arrays erfahren Sie im Abschnitt »Arrays«.

1.3.6 Resource-Typ

Der Resource-Typ bezeichnet eine spezielle Variable, die eine Referenz auf eine externe Ressource enthält. Resourcen werden von bestimmten Funktionen erzeugt und benutzt. Einige Ressourcen:

- COM – Referenz auf ein COM-Objekt
- ftp – FTP-Verbindung
- gd – GD-Grafik
- imap – Verbindung zu einem IMAP- oder POP3-Server herstellen
- msql query – mSQL-Ergebnis
- mysql result – MySQL-Ergebnis

- file – Datei-Handle
- xml – XML-Parser

Hinweis: Der Resource-Typ wurde in PHP 4 eingeführt.

Freigabe von Ressourcen

Aufgrund des Reference-Counting-Systems, das mit der Zend-Engine von PHP 4 eingeführt wurde, wird automatisch erkannt, wenn auf eine Ressource nicht mehr zugegriffen wird. Wenn dies der Fall ist, werden alle Ressourcen, die für diese Ressource in Gebrauch waren, durch den »Müllsammler« (Garbage Collector) freigegeben. Aus diesem Grund ist es nur in seltenen Fällen notwendig, Speicher manuell durch Aufruf von `free_result`-Funktionen freizugeben.

Achtung: Persistente Datenbankverbindungen stellen einen Sonderfall dar, sie werden durch den Garbage Collector nicht automatisch entfernt.

1.3.7 NULL

Das PHP-Schlüsselwort `NULL` ist ein besonderer Wert, der für »kein Wert« steht. Technisch gesehen ist `NULL` ein Wert des Typs `Objekt`. Wenn eine Variable diesen Wert besitzt, weiß man demnach, dass sie kein gültiges Objekt oder Array enthält. Zudem weiß man, dass sie auch weder eine Zahl, eine Zeichenkette, einen booleschen Wert noch eine Funktion enthält.

C/C++-Programmierer sollten auch beachten, dass `NULL` in PHP nicht dasselbe ist wie 0. Unter bestimmten Umständen wird `NULL` zwar in eine 0 umgewandelt, aber es liegt keine Äquivalenz vor.

Eine Variable wird als NULL interpretiert, wenn:

- ihr die Konstante `NULL` als Wert zugewiesen wurde,
- ihr bis jetzt kein Wert zugewiesen wurde,
- sie mit `unset()` gelöscht wurde.

Beispiel:

```
<?php
// Ausgane - Vorname ist NULL
if ($vorname == NULL) echo "Vorname ist NULL";
?>
```

Der Wert `NULL` kann in verschiedenen Situationen verwendet werden. Hier einige Beispiele für derartige Situationen:

- Eine Variable hat noch keinen Wert erhalten.
- Eine Variable enthält keinen Wert mehr.

- Eine Funktion kann keinen Wert zurückgeben, weil kein entsprechender Wert verfügbar ist; in diesem Fall wird der Nullwert zurückgegeben.

Hinweis: `NULL` ist der einzig mögliche Wert des Typs `NULL`.

1.3.8 Typen – Besonderheiten

Automatische Typenkonvertierung

Ein bedeutender Unterschied zwischen PHP und Sprachen wie C/C++ und Java liegt darin, dass PHP nicht typisiert ist. Das bedeutet, dass Variablenwerte beliebige Datentypen enthalten können. Im Gegensatz hierzu können Variablen in C/C++ und Java jeweils nur einen einzigen Datentyp enthalten.

Beispiel:

```
$zahl = 5;
$zahl = "fünf";
```

In PHP ist diese Zuweisung zulässig. Einer Variablen `zahl` wird zunächst eine Zahl und später eine Zeichenkette zugewiesen. In C/C++ oder Java wären diese Codezeilen unzulässig. Da PHP eine Sprache ohne explizite Typen ist, müssen Variablendeklarationen auch keinen Datentyp angeben, wie dies in C/C++ und Java der Fall ist. In diesen Sprachen deklariert man eine Variable, indem man den Namen des Datentyps angibt, den die Variable aufnehmen soll, und dahinter wird der Name der Variablen angegeben.

Beispiel:

```
// Deklaration einer Integer-Variablen in C/C++ oder Java
int zahl;
```

In PHP hingegen verwenden Sie für die Variablendeklaration lediglich einen gültigen Variablennamen und brauchen keinen Typ anzugeben.

Beispiel:

```
// Deklaration einer PHP-Variablen ohne Typ
$zahl = 100;
```

Eine Folge aus dem Nichtvorhandensein von Typen in PHP besteht darin, dass Werte automatisch zwischen verschiedenen Typen konvertiert werden können. Wenn man zum Beispiel versucht, eine Zahl an eine Zeichenkette anzuhängen, setzt PHP die Zahl automatisch in die entsprechende Zeichenkette um, die dann angehängt werden kann.

Beispiel:

```
<?php
// Variablen
$wort = "Besucher";
$nummer = 5;
$kombination = $nummer . $wort;
```

```
// Ausgabe - 5 Besucher
echo $kombination;
?>
```

Die Tatsache, dass PHP untypisiert ist, verleiht der Sprache die Flexibilität und Einfachheit, die für eine Skriptsprache wünschenswert ist. Im folgenden Abschnitt werden Sie die automatische Typkonvertierung genauer kennen lernen, da sie ein wesentlicher Bestandteil der Sprache ist.

Ein Beispiel für die automatische Typkonvertierung von PHP ist der +-Operator. Ist einer der zu addierenden Werte vom Typ `float`, werden alle Werte als `float`-Typ behandelt. Auch das Ergebnis der Addition wird vom Typ `float` sein. Andernfalls werden die Werte als `integer`-Typen angesehen und das Ergebnis wird ebenfalls vom Typ `integer` sein. Beachten Sie, dass hierdurch NICHT der Typ der Operanden selbst beeinflusst wird; der Unterschied liegt einzig und allein in der Auswertung dieser Operanden.

Beispiel:

```
<?php
// String
$wert = "0";
// Integer
$wert += 2;
// Ausgabe - 2 (Integer)
echo $wert;
?>
```

Hier eine Zusammenfassung zur automatischen Typzuweisung:

- In Anführungszeichen "" oder ' ' eingeschlossene Zeichen werden als String interpretiert.
- Eine Zahl ohne Punkt wird als Ganzzahl interpretiert.
- Eine Zahl mit Punkt wird als Fließkommazahl interpretiert.
- Bei der Auswertung von Ausdrücken bestimmt der verwendete Operator den Datentyp des Ergebnisses.

1.3.9 Typumwandlung

Explizite Typumwandlung

Typumwandlung in PHP funktioniert vielfach wie in C. Der Name des gewünschten Typs wird vor der umzuwandelnden Variablen in Klammern gesetzt, dies wird auch als `cast`-Operation bezeichnet.

Beispiel:

```
<?php
// Integer
$zahl = 100;
// Float
```

```
$zahl = (float) $zahl;
// Ausgabe - float(100)
echo var_dump($zahl);
?>
```

Folgende Umwandlungen sind möglich:

- (int), (integer) – nach integer
- (bool), (boolean) – nach boolean
- (float), (double), (real) – nach float
- (string) – nach string
- (array) – nach array
- (object) – Wandlung zum Objekt

Anstatt eine Variable in einen String umzuwandeln, können Sie die Variable auch in doppelte Anführungszeichen einschließen.

Beachten Sie, dass Tabulatoren und Leerzeichen innerhalb der Klammern erlaubt sind. Deshalb sind die folgenden Beispiele identisch:

```
$zahl = (int) $zahl;
$zahl = ( int ) $zahl;
```

Es ist nicht immer offenkundig, was bei der Typumwandlung geschieht. Sollten Sie eine Umwandlung eines Arrays zu einem String vornehmen oder erzwingen, ist das Ergebnis das Wort »Array«. Wenn Sie eine Umwandlung eines Objekts zu einem String vornehmen oder erzwingen, ist das Ergebnis das Wort »Objekt«.

Bei der Umwandlung einer skalaren oder String-Variablen zu einem Array wird die Variable das erste Element des Arrays:

```
<?php
$vorname = 'Caroline';
$personen = (array) $vorname;
// Ausgabe - Caroline
echo $personen[0];
?>
```

Sobald eine skalare oder String-Variable in ein Objekt gewandelt wird, wird die Variable zu einem Attribut des Objekts; der Eigenschaftsname wird `'scalar'`:

```
<?php
$vorname = 'Caroline';
$obj = (object) $vorname;
// Ausgabe - Caroline
echo $obj->scalar;
?>
```

Bei der Umwandlung sollten Sie berücksichtigen, dass nicht sämtliche Richtungen sinnvoll sind. In der folgenden Tabelle stellen wir Ihnen sinnvolle mögliche Kombinationen vor.

Zieltyp	*Sinnvolle Quelltypen*	*Unwandlungsprinzip*
integer	double	Dezimale werden abgeschnitten (keine Rundung).
int	string	Wird keine Zahl erkannt, wird 0 zurückgegeben.
double	integer	unverändert.
real	string	Wird keine Zahl erkannt, wird 0 zurückgegeben.
string	integer	Gibt die Zahl als Zeichenkette zurück.
	double	Gibt die Zahl als Zeichenkette zurück.
array	object	Wird direkt umgewandelt.
	integer	Es entsteht ein Array mit einem Element vom Ursprungstyp.
	string	Es entsteht ein Array mit einem Element vom Ursprungstyp.
	double	Es entsteht ein Array mit einem Element vom Ursprungstyp.
object	array	Wird direkt umgewandelt.
	integer	Es entsteht ein Objekt mit einer Eigenschaft, die durch die Variable repräsentiert wird.
	string	Es entsteht ein Objekt mit einer Eigenschaft, die durch die Variable repräsentiert wird.
	double	Es entsteht ein Objekt mit einer Eigenschaft, die durch die Variable repräsentiert wird.

Zeichenkettenkonvertierung

Bei der Umwandlung von Zeichenketten wendet PHP bestimmte Regeln an. Mit deren Kenntnis können Sie das Ergebnis voraussagen und von der Umwandlung sichern Gebrauch machen:

- `integer` entsteht, wenn die Zeichenkette mit einem gültigen numerischen Zeichen (Ziffer, Plus oder Minus) beginnt und dahinter nicht die Zeichen `.`, `e` oder `E` folgen. Ist der erste Teil der Zeichenkette ein gültiger Ausdruck, wird der Rest ignoriert.
- `float` (double) entsteht, wenn die Zeichenkette mit einem gültigen numerischen Zeichen (Ziffer, Plus oder Minus) beginnt und dahinter die Zeichen `.`, `e` oder `E` folgen. Ist der erste Teil der Zeichenkette ein gültiger Ausdruck, wird der Rest ignoriert.

Hinweis: Die Zeichen `e` oder `E` dienen der Darstellung von Exponenten. Die Schreibweise `Enn` steht für $x10^{nn}$. Generell dient der Punkt . als Dezimaltrennzeichen. Sie müssen dies bei Zuweisungen von Variablen berücksichtigen. Zur Ausgabe lassen sich Zahlen mit der Funktion `number_format` in die benötigte Form bringen (deutsche Schreibweise mit Komma).

Umwandlungsfunktion

Neben der Angabe des Datentyps als `cast`-Ausdruck kann auch die Funktion `settype()` eingesetzt werden. In manchen Ausdrücken wird eine Funktion erwartet, oft dient der Einsatz jedoch lediglich der besseren Lesbarkeit.

```
settype (string var, string type)
```

Der Typ der Variablen `var` wird festgelegt als `type`. Mögliche Werte für `type` sind:

- "integer"
- "double"
- "string"
- "array"
- "object"

Beispiel:

```
<?php
$preis = 9.99;
settype($preis,"integer");
// Ausgabe (9)
echo $preis
?>
```

Beispiel:

```
<?php
$preis = 9.99;
settype($preis,"object");
// Ausgabe Object id #1
echo $preis;
// Ausgabe (9.99)
echo $preis->scalar;
?>
```

Hinweis: Bei erfolgreicher Umwandlung liefert `settype()` TRUE, sonst FALSE. So haben Sie die Möglichkeit, auf eine fehlerhafte Umwandlung mit Hilfe einer Bedingung zu reagieren.

Beispiel:

```
<?php
$preis = 9.99;
// Aushabe (9)
if (settype($preis,"integer") == 1) {
       echo $preis;
} else {
       echo "Fehler!";
}
?>
```

Sollte Ihnen `settype()` zu umständlich sein, können Sie auch die abgeleiteten `val`-Funktionen verwenden:

- `intval(string var)` – Diese Funktion wandelt in `integer` um.
- `doubleval (string var)` – Diese Funktion wandelt in `double` um.
- `strval(string var)` – Diese Funktion wandelt in `string` um.

Beispiel:

```
<?php
$preis = 9.99;
// Aushabe (9)
echo intval($preis);
?>
```

1.3.10 Datentypen bestimmen

Einsatz von var_dump()

Um beispielsweise den Typ und den Wert eines bestimmten Ausdrucks (Expression) zu überprüfen, können Sie `var_dump()` einsetzen.

Beispiel:

```
<pre>
<?php
$personen = array(array('Matthias','Kannengiesser',29),
                  array('Caroline','Kannengiesser',25),
                  array('Gülten','Kannengiesser', 59,
                          array('Eltern','Mutter')));
var_dump($personen);
?>
</pre>
```

Ausgabe:

```
array(3) {
  [0]=>
  array(3) {
    [0]=>
    string(8) "Matthias"
    [1]=>
    string(13) "Kannengiesser"
    [2]=>
    int(29)
  }
  [1]=>
  array(3) {
    [0]=>
    string(8) "Caroline"
    [1]=>
    string(13) "Kannengiesser"
```

```
    [2]=>
    int(25)
  }
  [2]=>
  array(4) {
    [0]=>
    string(6) "Gülten"
    [1]=>
    string(13) "Kannengiesser"
    [2]=>
    int(59)
    [3]=>
    array(2) {
      [0]=>
      string(6) "Eltern"
      [1]=>
      string(6) "Mutter"
    }
  }
}
```

Diese Funktion hat die Aufgabe, Informationen über Typ und Wert des Parameters zurückzugeben. Arrays und Objekte werden rekursiv, von innen nach außen, durchlaufen und mit entsprechender Einrückung dargestellt.

Sollten Sie zur Fehlersuche lediglich eine lesbare Darstellung eines Typs benötigen, steht Ihnen hierfür `gettype()` zur Verfügung.

Einsatz von gettype()

Beispiel:

```
<?php
$vorname = "Caroline";
$alter = 25;

// Ausgabe - string
echo gettype($vorname);
// Ausgabe - integer
echo gettype($alter);
?>
```

Die Funktion gibt die bereits bekannten Typbezeichner als Zeichenkette zurück. Konnte der Typ nicht erkannt werden, so wird die Zeichenkette `unknown type` erzeugt.

> **Achtung:** Um den Typ zu prüfen, sollten Sie nicht `gettype()` verwenden. Stattdessen sollten Sie die `is_type`-Funktionen verwenden. Diese finden sie weiter unten im Abschnitt »Datentypen bestimmen«.

Bei logischen Ausdrücken ist die Verwendung von `gettype()` zu umständlich. Sie können daher eine ganze Reihe von `is_type`-Funktionen einsetzen, die `True (1)` oder `False (0)` zurückgeben.

Funktion	*Beschreibung*

`is_long(string var)`
`is_integer(string var)`
`is_int(string var)`
	Ermittelt, ob es sich um einen Ausdruck vom Typ `integer` handelt. Gibt 1 zurück, wenn die Variable vom Typ `integer` ist.

`is_double(string var)`
`is_real(string var)`
`is_float(string var)`
	Ermittelt, ob es sich um einen Ausdruck vom Typ `double` bzw. `float` handelt. Gibt 1 zurück, wenn die Variable vom Typ `double` oder `float` ist.

`is_string(string var)`	Ermittelt, ob es sich um einen Ausdruck vom Typ `string` handelt. Gibt 1 zurück, wenn die Variable vom Typ `string` ist.
`is_numeric(string var)`	Ermittelt, ob es sich um einen numerischen Typ (`integer`, `double`) handelt. Gibt 1 zurück, wenn die Variable vom Typ `integer` oder `double` ist.
`is_bool(string var)`	Ermittelt, ob es sich um einen Ausdruck vom Typ `boolean` handelt. Gibt 1 zurück, wenn die Variable vom Typ `boolean` ist.
`is_array(string var)`	Ermittelt, ob es sich um einen Ausdruck vom Typ `array` handelt. Gibt 1 zurück, wenn die Variable vom Typ `array` ist.
`is_object(string var)`	Ermittelt, ob es sich um einen Ausdruck vom Typ `object` bzw. eine Objektvariable handelt. Gibt 1 zurück, wenn die Variable vom Typ `object` ist.
`is_null`	Ermittelt, ob es sich um einen Ausdruck vom Typ `null` handelt. Gibt 1 zurück, wenn die Variable vom Typ `null` ist.
`is_resource`	Ermittelt, ob es sich um einen Ausdruck vom Typ `resource` handelt. Gibt 1 zurück, wenn die Variable vom Typ `resource` ist.
`is_scalar`	Ermittelt, ob es sich um einen Ausdruck vom Typ `integer`, `float`, `string` oder `boolean` handelt. Gibt 1 zurück, wenn die Variable vom Typ `integer`, `float`, `string` oder `boolean` ist.

Beispiel:

```
<?php
$preis = "9.99";
// Aushabe - Es ist ein String
if (is_string($preis)) {
       echo "Es ist ein String";
} else {
       echo "Kein String";
}
?>
```

Beispiel:

```
<?php
$signal = FALSE;
// Aushabe - Ist bool
if (is_bool($signal)) {
       echo "Ist bool";
} else {
       echo "Kein bool";
}
?>
```

Allerdings verhalten sich die Funktionen nicht alle gleich. So akzeptiert die Funktion `is_numeric()` auch numerische Werte, die in Anführungszeichen, also als Strings übergeben werden. Der Wert darf lediglich keine Buchstaben enthalten.

Beispiel:

```
<?php
$wert = "9.99";
// Aushabe - Ist eine Zahl
if (is_numeric($wert)) {
        echo "Ist eine Zahl";
} else {
        echo "Ist keine Zahl";
}
?>
```

Wenn Sie die Umwandlung in einen bestimmten Typ erzwingen wollen, erreichen Sie dies entweder durch `cast`-Ausdrücke oder durch Gebrauch der Funktion `settype()`.

Beachten Sie, dass sich eine Variable in bestimmten Situationen unterschiedlich verhalten kann, abhängig vom Typ, dem die Variable zum Zeitpunkt ihrer Verarbeitung entspricht.

1.4 Variablen

1.4.1 Was ist eine Variable?

Eine der wichtigsten Merkmale einer Programmiersprache ist die Fähigkeit, Daten zu verwalten. Diese Daten werden in Variablen gespeichert und können aus diesen auch wieder ausgelesen werden. Eine Variable kann man sich wie einen Behälter für Informationen vorstellen.

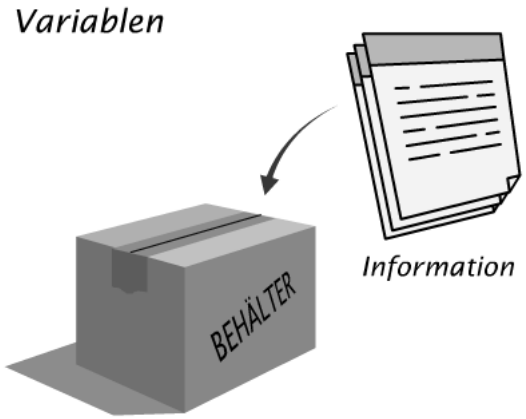

Bild 1.1: Variable – ein Behälter, der Daten bzw. Informationen enthält

Eine Variable ist

- ein Lager mit einer eindeutigen Kennung, in das man Dinge ablegen und herausnehmen kann;
- eine Einkaufstüte, in die Sie Waren hineinlegen und herausnehmen können;
- ein Notizzettel, auf dem Sie Termine eintragen, entfernen oder ändern können.

Dabei müssen Sie sich eines klar machen: Der Behälter an sich bleibt immer gleich, lediglich sein Inhalt kann sich ändern. Machen Sie sich aber keine Sorgen, Sie können theoretisch so viele Variablen erzeugen, wie es Ihr Arbeitsspeicher zulässt.

Hinweis: Variablen findet man in praktisch jeder Programmiersprache, sei es C/C++, Java, JavaScript oder PHP. Sie dienen dazu, Daten zu repräsentieren, die sich im Laufe eines Programms verändern können. Das Gegenstück zu den Variablen sind die Konstanten, deren Wert sich nicht ändert.

Es empfiehlt sich, einer Variablen beim Definieren stets einen bekannten Wert zuzuweisen. Dies wird als Initialisieren einer Variablen bezeichnet. Die Initialisierung ermöglicht es Ihnen, den Wert der Variablen zu überwachen und Veränderungen zu erfassen.

Der Datentyp des Werts einer Variablen bestimmt, wie dieser sich bei einer Zuweisung in einem Skript ändert. Zu den am häufigsten in Variablen gespeicherten Informationen gehören:

- Nutzerinformation (wie Nutzernamen, Anschrift, Telefonnumer etc.)
- Ergebnisse aus mathematischen Berechnungen

1.4.2 Variablendefinition

Variablen bestehen aus drei Teilen:

- dem Variablennamen,
- einem Speicherbereich,
- einem Wert.

Um in einer Variablen einen Wert ablegen zu können, muss es irgendein Medium geben, in dem der Wert der Variablen festgehalten werden kann. Wären Variablen Notizzettel, wäre Papier das gesuchte Medium. Da Variablen jedoch im Computer existieren, scheidet das Medium Papier aus.

Wo werden die Werte der Variablen gespeichert?

Natürlich werden Werte im Arbeitsspeicher gespeichtert, genauer gesagt im Arbeitsspeicher des Computers, auf dem Serverrechner, auf dem der PHP-Code ausgeführt wird.

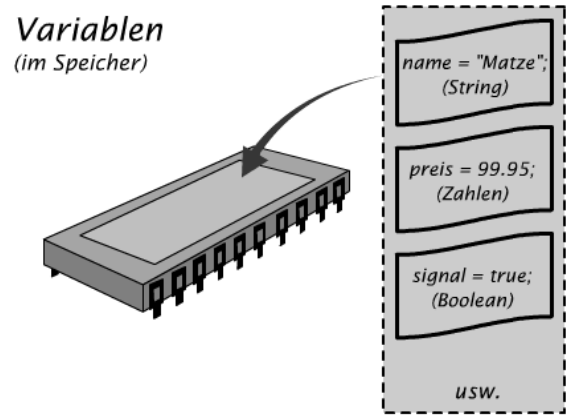

Bild 1.2: Variablen im Arbeitsspeicher

Sie sollten sich dabei folgendes Szenario vor Augen halten: Der PHP-Code, den wir erstellen, wird erst beim Aufruf des PHP-Dokuments ausgeführt. Trifft nun der PHP-Interpreter dabei auf eine Variablendefinition, reserviert er im Arbeitsspeicher einen passenden Speicherbereich und verbindet ihn mit der Variablen. Wird im PHP-Code der Wert der Variablen abgefragt, liest der Interpreter den aktuellen Wert aus, der im Speicherbereich der Variablen abgelegt ist, und liefert ihn zurück. Wird der Variablen ein neuer Wert zugewiesen, schreibt der Interpreter den neuen Wert in den Speicherbereich der Variablen, dabei wird der alte Wert gelöscht.

Die Besonderheit liegt darin, dass die gesamte Speicherverwaltung vom Interpreter übernommen wird. Sie brauchen sich als Programmierer nicht weiter darum kümmern, in welchen Speicherbereich die Variable abgelegt wurde. Ein Programmierer sollte wissen, dass hinter einer Variablen ein Speicherbereich steht, doch für die tägliche Arbeit genügt es zu wissen, dass Variablen über ihre Namen angesprochen werden. Dies hat auch Auswirkungen auf die gängige Sprachregelung.

Beispiel:

```
$vorname = "Matthias";
```

Für den Interpreter bedeutet diese Anweisung:

»Nimm die Zeichenkette `Matthias` und schreibe sie in den Speicherbereich, der für die Variable mit Namen `$vorname` reserviert wurde.«

Wir sagen dazu einfach:

»Der Variablen `$vorname` wird die Zeichenkette Matthias zugewiesen.«

1.4.3 L- und R-Wert

Wie Sie wissen, sind Variablen mit Speicherbereichen verbunden. Über den Variablennamen können Sie sowohl einen Wert in einer Variablen speichern als auch den Wert

einer Variablen auslesen. Woher weiß der Interpreter, ob Sie den Wert einer Variablen auslesen oder einen neuen Wert in einer Variablen speichern wollen?

Das ist im Grunde recht einfach. Nach der Definition repräsentiert der Variablenname grundsätzlich den Wert der Variablen. Nur wenn der Variablenname auf der linken Seite einer Zuweisung auftaucht, steht er für den Speicherbereich, in dem der Inhaltswert des rechten Teils der Zuweisung abzulegen ist.

In einer einzigen Zuweisung kann ein Variablenname also sowohl den Wert der Variablen als auch den Speicherbereich der Variablen repräsentieren.

Beispiel:

```
zahlEins = zahlEins + 10;
```

Aus diesem Grund spricht man auch vom

- L-Wert (Linkswert = Speicherbereich der Variablen) und
- R-Wert (Rechtswert = akueller Wert der Variablen).

Sie sollten sich diesen Zusammenhang besonders gut merken, da Sie immer wieder damit konfrontiert werden.

1.4.4 Benennen von Variablen

Wie Sie bereits wissen, besitzt jede Variable einen Namen oder *Bezeichner*. Bei der Benennung von Variablen sind einige Regeln zu beachten:

- Bezeichner müssen immer mit einem Dollarzeichen *($)* beginnen.
- Die darauf folgenden Zeichen können beliebige Buchstaben, Ziffern oder Unterstriche sein.
- Zwischen Groß- und Kleinschreibung wird unterschieden.
- Es dürfen keine von PHP reservierten Wörter als Bezeichner verwendet werden.
- Bezeichner müssen in einer Zeile Platz finden.
- Bezeichner dürfen keine Leerzeichen enthalten.

Achtung: Der Variablenname sollte auch nie mehr als 255 Zeichen besitzen.

Gültige Variablennamen

```
$einheit = "Meter";
$zahl = 1000;
$_bilder = "foto.jpg";
$spielerName = "Thomas";
$max_wert = 100;
```

Ungültige Variablennamen

```
1einheit = "Meter";
%zahl = "100%";
$ = "Dollarzeichen";
_ = "Unterstrich";
max wert = 100;
wert: = "Zehntausend";
```

1.4.5 Variablenwerte

Wenn Sie eine oder mehrere Variablen deklariert haben, stellt sich die Frage, was Sie mit den Variablen anfangen können. Der erste Schritt wäre, der Variablen einen Wert zuzuweisen.

Wie man einer Variablen einen Wert zuweist, haben Sie bereits an zahlreichen Beispielen im Buch kennen gelernt. Der Variablen wird der neue Wert mit Hilfe des (=)-Operators zugewiesen.

Beispiel:

```
$artikelnummer = 12568;
$firmenname = "Madania Netware";
```

Die Syntax ist nicht schwierig, die Variable ist folgendermaßen aufgebaut:

- Links steht die Variable, der ein neuer Wert zugewiesen wird.
- Es folgt der Zuweisungsoperator (=)-Operator.
- Rechts vom Zuweisungsoperator steht der neue Wert der Variablen.
- Abgeschlossen wird die Zeile mit einem Semikolon.

Hinweis: Beim (=)-Operator handelt es sich um einen Zuweisungsoperator. Der Operator sorgt dafür, dass der Variablen ein neuer Wert zugewiesen wird. Ein Vergleich findet hier nicht statt. Natürlich kann auch auf der rechten Seite des Zuweisungsoperators ein Ausdruck stehen, den der Interpreter zu einem einzelnen Wert umberechnen kann.

Beispiel:

```
$summe = 10 + 10;      // Die Variable $summe enthält den Wert 20
```

In einem solchen Fall berechnet der Interpreter zuerst den Ausdruck auf der rechten Seite. Das Ergebnis der Berechnung weist er als neuen Wert der Variablen `$summe` zu. Die Variable hat danach den Wert 20. Nun kennen Sie auch die interne Arbeitsweise des Interpreters. Sie sollten nun versuchen, den gespeicherten Wert abzurufen. Denn eine Variable samt ihrem Wert ist nur dann interessant, wenn Sie den Wert der Variablen zu einem späteren Zeitpunkt abrufen können, beispielsweise um deren Wert einer anderen Variablen zuzuweisen oder um ihn in die Berechnung einer Formel einfließen zu lassen.

Beispiel:

```
$zahleins = 25;
$zahlzwei = 75;

$zahleins = $zahlzwei;
```

Hier wird der Variablen `$zahleins` der Wert der Variablen `$zahlzwei` zugewiesen. Danach enthalten beide Variablen den Wert 75.

Obwohl es in der Programmierung relativ häufig vorkommt, dass man einer Variablen den Wert einer anderen Variablen zuweist, ist es meist interessanter, wenn auf der Grundlage des einen Wertes ein neuer Wert berechnet wird.

Beispiel:

```
<?php
// Prozentberechnung (Anteil in Prozent)
$gesamt = 1000;
$aktanteil = 100;

// Wert 10 (entspricht 10%)
$prozentanteil = ($aktanteil * 100)/$gesamt;
// Ausgabe (10)
echo $prozentanteil;
?>
```

Hier wird der Wert der Variablen `$aktanteil` mit Hilfe des Multiplikations-Operators mit dem Faktor 100 multipliziert und anschließend durch den Wert der Variablen `gesamt` mit Hilfe des Divisions-Operators dividiert. Das Ergebnis wird in `$prozentanteil` gespeichert. Beachten Sie, dass die Werte der Variablen `$aktanteil` und `gesamt` dabei nicht verändert werden. Sie werden lediglich abgefragt und fließen in die Berechnung ein.

Variablenwerte ohne Hilfe von temporären Variablen austauschen

Sollten Sie die Werte zweier Variablen vertauschen wollen, ohne dabei zusätzliche Variablen in Anspruch zu nehmen, können Sie die Funktion `list()` verwenden.

Beispiel:

```
<?php
$wert1 = 100;
$wert2 = 999;
list($wert1,$wert2) = array($wert2,$wert1);
// Ausgabe (999)
echo "$wert1<br>";
// Ausgabe (100)
echo "$wert2<br>";
?>
```

Wie Sie sehen, werden den Elementen eines Arrays einzelne Variablen zugewiesen. Das Gegenstück auf der rechten Seite des Ausdrucks `array()` ermöglicht Ihnen, Arrays aus einzelnen Werten zu bilden. Indem Sie das Array, das von `array()` zurückgegeben wird,

den Variablen in `list()` zuweisen, verändern Sie die Anordnung dieser Werte. Dies funktioniert im Übrigen auch mit mehr als zwei Variablen.

Beispiel:

```
<?php
$wert1 = 100;
$wert2 = 999;
$wert3 = 10000;
list($wert1,$wert2,$wert3) = array($wert3,$wert2,$wert1);
// Ausgabe (10000)
echo "$wert1<br>";
// Ausgabe (999)
echo "$wert2<br>";
// Ausgabe (100)
echo "$wert3<br>";
?>
```

Achtung: Diese Methode ist nicht schneller als der Einsatz von temporären Variablen. Sie sollten sie aus Gründen der Lesbarkeit, nicht der Geschwindigkeit einsetzen.

Woher können die Daten einer Variablen noch kommen?

Hier einige mögliche Quellen:

Im einfachsten Fall sind es vordefinierte konstante Werte (Konstanten).

Beispiel:

```
<?php
$version = PHP_VERSION;
$betrieb_os = PHP_OS;

// Ausgabe 5.0.0RC1-dev
echo $version;
// Ausgabe WIN32
echo $betrieb_os;
?>
```

Sie rufen eine Funktion bzw. Methode auf, die einen Ergebniswert zurückliefert.

Beispiel:

```
<?php
$wert = abs(-10);
// Ausgabe (10)
echo $wert;
?>
```

1.4.6 Unwandeln und Prüfen von Variablen

PHP stellt eine Vielzahl von Funktionen zur Verarbeitung von Variablen zur Verfügung, so dass es dem Entwickler nicht schwer fallen wird, sie im Griff zu haben.

Einsatz von isset()

Bei logischen Vergleichen kann es äußerst wichtig sein zu prüfen, ob eine Variable mit 0 oder einem leeren String gefüllt wurde oder überhaupt noch nicht zugewiesen wurde. Sie können mit der Funktion `isset()` testen, ob eine Variable existiert.

Beispiel:

```
<?php
$preis = 9.99;
if (isset($preis)) {
       echo $preis;
} else {
       echo "Umsonst!";
}
?>
```

Ausgabe:

```
9.99
```

Wie man sieht, lässt sich die Funktion hervorragend in einer Bedingung unterbringen und somit jederzeit überprüfen, ob die jeweilige Variable bereits existiert oder nicht.

Einsatz von empty()

Wenn Sie nur testen möchten, ob ein Wert zugewiesen wurde, ist die Funktion `empty()` genau das Richtige.

Beispiel:

```
<?php
$vorname = "";
if (empty($vorname)) {
       echo "Ist leer und existiert";
} else {
       echo "Ist nicht leer und existiert";
}
?>
```

Ausgabe:

```
Ist leer und existiert
```

Die Funktion gibt `1` (`True`) zurück, wenn der Inhalt der Variablen 0 oder ein leeren String ist und die Variable existiert.

Einsatz von unset()

Sollten Sie vorhaben, eine Zuweisung bzw. Variable wieder aufzuheben und die Variable damit zu löschen, steht Ihnen die Funktion `unset()` zur Verfügung.

Beispiel:

```
<?php
$vorname = "Caroline";
unset($vorname);
if (isset($vorname)) {
        echo "Existiert!";
} else {
        echo "Existiert Nicht!";
}
?>
```

Ausgabe:

```
Existiert Nicht!
```

Hier eine Aufstellung der wichtigsten Funktion zum Prüfen von Variablen samt ihren Rückgabewerten.

Funktion	*Beschreibung*	*Rückgabewert*
is_integer	Prüft, ob die Variable vom Typ `integer` ist.	TRUE (1)/FALSE (0)
is_int	Prüft, ob die Variable vom Typ `integer` ist.	TRUE (1)/FALSE (0)
is_long	Prüft, ob die Variable vom Typ `integer` ist.	TRUE (1)/FALSE (0)
is_real	Prüft, ob die Variable vom Typ `real` ist.	TRUE (1)/FALSE (0)
is_double	Prüft, ob die Variable vom Typ `double` ist.	TRUE (1)/FALSE (0)
is_float	Prüft, ob die Variable vom Typ `float` ist.	TRUE (1)/FALSE (0)
is_bool	Prüft, ob die Variable vom Typ `bool` ist.	TRUE (1)/FALSE (0)
is_array	Prüft, ob die Variable vom Typ `array` ist.	TRUE (1)/FALSE (0)
is_object	Prüft, ob die Variable vom Typ `object` ist.	TRUE (1)/FALSE (0)
is_string	Prüft, ob die Variable vom Typ `string` ist.	TRUE (1)/FALSE (0)
is_null	Prüft, ob die Variable vom Typ `null` ist.	TRUE (1)/FALSE (0)
is_numeric	Prüft, ob die Variable eine Zahl oder ein numerischer String ist.	TRUE (1)/FALSE (0)
is_resource	Prüft, ob die Variable eine Ressource ist.	TRUE (1)/FALSE (0)
is_scalar	Prüft, ob die Variable vom Typ `integer`, `float`, `string` oder `boolean` ist.	TRUE (1)/FALSE (0)
isset	Prüft, ob die Variable definiert ist.	TRUE (1)/FALSE (0)
unset	Löscht eine Variable aus dem Speicher.	1
empty	Prüft, ob die Variable »leer« ist.	TRUE (1)/FALSE (0)
intval	Wandelt den Typ einer Variablen in `integer` um.	integer
strval	Wandelt den Typ einer Variablen in `string` um.	string
doubleval	Wandelt den Typ einer Variablen in `double` um.	double
floatval	Wandelt den Typ einer Variablen in `float` um.	float
gettype	Ermittelt den Typ der Variablen.	string
settype	Legt den Typ der Variablen fest.	TRUE (1)/FALSE (0)

Einsatz von get_defined_vars()

Dies Funktion `get_defined_vars()` gibt ein mehrdimensionales Array zurück, welches eine Liste sämtlicher definierter Variablen enthält. Dabei handelt es sich um Variablen aus der Entwicklungsumgebung, vom Server oder um benutzerdefinierte Variablen.

Beispiel – Servervariablen:

```
<pre>
<?php
$liste = get_defined_vars();
print_r($liste["_SERVER"]);
?>
</pre>
```

Ausgabe:

```
Array
(
    [HTTP_ACCEPT] => */*
    [HTTP_ACCEPT_LANGUAGE] => de
    [HTTP_ACCEPT_ENCODING] => gzip, deflate
    [HTTP_USER_AGENT] => Mozilla/4.0…
    [HTTP_HOST] => localhost
    [HTTP_CONNECTION] => Keep-Alive
    [HTTP_CACHE_CONTROL] => no-cache
    [PATH] => C:\WINDOWS;C:\WINDOWS\COMMAND
    [COMSPEC] => C:\WINDOWS\COMMAND.COM
    [WINDIR] => C:\WINDOWS
    [SERVER_SIGNATURE] => Apache/2.0.48 (Win32) PHP/5.0…
    [SERVER_SOFTWARE] => Apache/2.0.48 (Win32) PHP/5.0…
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => 127.0.0.1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => 127.0.0.1
    [DOCUMENT_ROOT] => C:/php5xampp-dev/htdocs
    [SERVER_ADMIN] => admin@localhost
    [SCRIPT_FILENAME] => C:/php5xampp-dev/htdocs/test.php
    [REMOTE_PORT] => 1334
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /php5/test.php
    [SCRIPT_NAME] => /php5/test.php
    [PHP_SELF] => /php5/test.php
)
```

Beispiel – Array Schlüssel des Systems:

```
<pre>
<?php
print_r(array_keys(get_defined_vars()));
?>
</pre>
```

Ausgabe:

```
Array
(
    [0] => GLOBALS
    [1] => HTTP_ACCEPT
    [2] => HTTP_ACCEPT_LANGUAGE
    [3] => HTTP_ACCEPT_ENCODING
    [4] => HTTP_USER_AGENT
    [5] => HTTP_HOST
    [6] => HTTP_CONNECTION
    [7] => HTTP_CACHE_CONTROL
    [8] => PATH
    [9] => COMSPEC
    [10] => WINDIR
    [11] => SERVER_SIGNATURE
    [12] => SERVER_SOFTWARE
    [13] => SERVER_NAME
    [14] => SERVER_ADDR
    [15] => SERVER_PORT
    [16] => REMOTE_ADDR
    [17] => DOCUMENT_ROOT
    [18] => SERVER_ADMIN
    [19] => SCRIPT_FILENAME
    [20] => REMOTE_PORT
    [21] => GATEWAY_INTERFACE
    [22] => SERVER_PROTOCOL
    [23] => REQUEST_METHOD
    [24] => QUERY_STRING
    [25] => REQUEST_URI
    [26] => SCRIPT_NAME
    [27] => PHP_SELF
    [28] => _POST
    [29] => HTTP_POST_VARS
    [30] => _GET
    [31] => HTTP_GET_VARS
    [32] => _COOKIE
    [33] => HTTP_COOKIE_VARS
    [34] => _SERVER
    [35] => HTTP_SERVER_VARS
    [36] => _ENV
    [37] => HTTP_ENV_VARS
    [38] => _FILES
    [39] => HTTP_POST_FILES
    [40] => _REQUEST
)
```

Beispiel:

```
<pre>
<?php
$liste = get_defined_vars();
print_r($liste);
?>
</pre>
```

Ausgabe:

Die Ausgabe haben wir uns und Ihnen erspart, nur so viel sei verraten: Sie erfahren dabei interessante Dinge über Ihren Server.

1.4.7 Gültigkeitsbereiche und Sichtbarkeit von Variablen

Ein wichtiges Thema im Zusammenhang mit Variablen ist deren Gültigkeitsbereich und Sichtbarkeit. Generell gilt, dass in PHP Variablen immer nur in ihrem lokalen Kontext sichtbar sind.

Dieser beinhaltet auch den Bereich für Dateien, die per `include`- oder `require`-Anweisung eingebunden wurden, z.B.:

```
$autor = "Matthias";
include "buch.inc.php";
```

Die Variable `$autor` ist auch in der eingebundenen Datei `buch.inc.php` verfügbar. Eine innerhalb einer Funktion definierte Variable ist außerhalb der Funktion nicht sichtbar. Umgekehrt gilt dasselbe, d.h., eine außerhalb sämtlicher Funktionsblöcke global definierte Variable hat innerhalb eines Funktionsblocks keine Gültigkeit.

Beispiel:

```
<?php
// Globaler Bereich
$preis = 9.99;
function berechne() {
       // Referenz auf einen lokalen Bereich
       return $preis;
}
$betrag = berechne();
if ($betrag) {
       echo $betrag;
} else {
       echo "Keine Ausgabe, die Variable ist lokal nicht sichtbar!";
}
?>
```

Ausgabe:

```
Keine Ausgabe, die Variable ist lokal nicht sichtbar!
```

Um zu erreichen, dass die globale Variable `$preis` auch lokal innerhalb der Funktion `berechne` bekannt ist, muss diese explizit mithilfe des Schlüsselworts `global` innerhalb der Funktion bekannt gemacht werden, man lädt sie sozusagen ein.

Beispiel:

```
<?php
$preis = 9.99;
function berechne() {
       global $preis;
```

```
        return $preis;
}
$betrag = berechne();
if ($betrag) {
        echo $betrag;
} else {
        echo "Keine Ausgabe, die Variable ist lokal nicht sichtbar!";
}
?>
```

Ausgabe:

```
9.99
```

Hinweis: Auf den Einsatz und die Verwendung von Funktionen wird im Abschnitt »Funktionen und Prozeduren« eingegangen.

Zugriff über $GLOBALS

Eine andere Möglichkeit, im lokalen Kontext einer Funktion auf eine globale Variable zuzugreifen, steht über das von PHP definierte Array `$GLOBALS` zur Verfügung.

Dabei handelt es sich um ein assoziatives Array, das die Namen der globalen Variablen als Schlüsselwörter verwendet. Den Zugriff innerhalb des lokalen Kontextes einer Funktion über das Array `$GLOBALS` auf die lokale Variable `$preis` zeigt folgendes Beispiel:

```
<?php
$preis = 9.99;
function berechne() {
        return $GLOBALS[preis];
}
echo berechne();
?>
```

Ausgabe:

```
9.99
```

Speicherklassen von Variablen

Eine weitere wichtige Eigenschaft von Variablen ist deren Speicherklasse. Normale Variablen verlieren beim Verlassen des lokalen Kontextes ihren Wert und werden beim Wiedereintritt neu initialisiert.

Sollen Variablen auch nach dem Verlassen eines Funktionsblocks ihren Wert behalten, müssen sie mithilfe des Schlüsselworts `static` als statische Variablen vereinbart werden.

Im folgenden Beispiel hat die Variable `$zaehler` keine statische Lebensdauer, sie wird bei jedem Neueintritt in den Funktionsblock erneut initialisiert. Das Ergebnis der Ausgabe bleibt trotzt zweimaligen Aufrufs der Funktion `setzeZaehler()` 1.

```
<?php
function setzeZaehler() {
       $zaehler = 0;
       $zaehler++;
       return $zaehler;
}
$zustand1 = setzeZaehler();
// Ausgabe (1)
echo $zustand1;
$zustand2 = setzeZaehler();
// Ausgabe (1)
echo $zustand2;
?>
```

Wird die Variable `$zaehler` jedoch statisch vereinbar, behält sie auch nach dem Verlassen des Funktionsblocks und dem Wiedereintritt bei einem erneuten Aufruf der Funktion `setzeZaehler()` ihren Wert, so dass sich, wie das folgende Beispiel zeigt, ein anderes Ergebnis zeigt:

```
<?php
function setzeZaehler() {
       static $zaehler = 0;
       $zaehler++;
       return $zaehler;
}
$zustand1 = setzeZaehler();
// Ausgabe (1)
echo $zustand1;
$zustand2 = setzeZaehler();
// Ausgabe (2)
echo $zustand2;
?>
```

Eine weitere interessante Anwendung in diesem Zusammenhang sind rekursive Funktionen. Das sind Funktionen, die sich aus sich selbst heraus aufrufen. Dabei besteht die Gefahr, so genannte Endlosschleifen bzw. Endlos-Rekursionen zu programmieren, welche die Performance des jeweiligen Systems äußerst negativ beeinflussen. Sie müssen also einen Weg vorsehen, eine solche Rekursion zu beenden. Die folgende Funktion `setzeZaehler()` zählt rekursiv bis 10. Die statische Variable `$zaehler` wird benutzt, um die Rekursion zu beenden:

```
<?php
function setzeZaehler() {
    static $zaehler = 0;

    $zaehler++;
    echo $zaehler;
    if ($zaehler < 10) {
        setzeZaehler();
    }
}
// Ausgabe (12345678910)
setzeZaehler();
?>
```

Achtung: Sollten Sie keine statische Variable verwenden, kommt es zu einer Endlosschleife.

1.4.8 Dynamische Variablen

Interessant ist die Möglichkeit in PHP, auch Variablennamen selbst in Variablen zu speichern und so quasi auf Variablen zuzugreifen. Die Zuweisung erfolgt in zwei Schritten. Zuerst wird eine normale Variable erzeugt, der Sie den Namen der dynamischen Variablen zuordnen:

```
$varname = "meinevariable";
```

Eine dynamische Variable nimmt den Wert einer Variablen als Namen. Der implizierten Variablen können Sie einen Wert zuweisen, indem Sie ihr zwei `$$`-Zeichen voranstellen:

```
$$varname = "PHP";
```

Sollten Sie die zuvor gezeigte Definition verwendet haben, gibt das Skript nun mit `echo($meinevariable)` den Wert »PHP« aus. Einmal erfolgte Zuweisungen bleiben von späteren Umbenennungen der führenden Variablen unberührt. Das folgende Beispiel zeigt einige Varianten:

```
<?php
$varname = "meinevariable";
echo "Variable varname ist: $varname <BR>";
$name = "PHP";
echo "Variable name ist: $name <BR>";
$$varname = "Dynamisch";
echo "Variable varname ist: $meinevariable <BR>";
$varname = "Programmieren";
echo "Variable varname ist: $meinevariable <BR>";
echo "Variable varname ist: $varname <BR>";
?>
```

Ausgabe:

```
Variable varname ist: meinevariable
Variable name ist: PHP
Variable varname ist: Dynamisch
Variable varname ist: Dynamisch
Variable varname ist: Programmieren
```

Sie sollten sich dieses Ergebnis in Ruhe betrachten.

Dynamische Variablen mit Arrays

Dynamische Variablen können auch mit Arrays genutzt werden. In diesem Fall kann es zu Zuordnungsproblemen kommen. Der PHP-Interpreter kann nicht immer eindeutig erkennen, auf welchen Teil der Konstruktion sich die Indizes beziehen. So könnte mit `$$zahl[1]` sowohl `$zahl[1]` eine Variable sein oder `$$zahl` als Teil eines Arrays mit dem Index 1. In solchen Fällen fassen Sie die logisch zusammenhängenden Teile des

Ausdrucks mit geschweiften Klammern zusammen, entweder `${$zahl[1]}` oder `${$zahl}[1]`.

Anwendungsmöglichkeiten

Diese komplizierte Art der Verarbeitung von Variablen mag auf den ersten Blick wenig sinnvoll erscheinen. Das folgende Beispiel zeitig, wie es angewendet werden kann.

Angenommen, Sie möchten sich Variablen über mehrere Skripts hinweg merken. Das kann in einer Datei oder einer Datenbank geschehen. In jedem Fall speichern Sie die zu speichernden Variablen in einem Array, wobei immer der Name der Variablen und der Inhalt zusammen ein Element ergeben.

Beispiel:

```
<?php
$personen = array(
        "vorname" => "Caroline",
        "nachname" => "Kannengiesser",
        "ort" => "Berlin",
        "alter" => 25
        );
if (is_array($personen)) {
        while (list($name,$wert) = each($personen)) {
                ${$name} = $wert;
                echo "$name : $name / ${$name} / $wert <BR>";
        }
}
?>
```

Ausgabe:

```
vorname : vorname / Caroline / Caroline
nachname : nachname / Kannengiesser / Kannengiesser
ort : ort / Berlin / Berlin
alter : alter / 25 / 25
```

Tipp: Betrachten Sie die Ausgabe und ziehen Sie Ihren eigenen Schluss!

Wenn dieses Skript abläuft, entstehen vier Variablen: `$vorname`, `$nachname`, `$ort` und `$alter`. Diese sind mit den Daten `"Caroline"`, `"Kannengiesser"`, `25` und `"Berlin"` belegt. Stellen Sie sich vor, anstatt des Arrays eine serialisierte Zeichenkette zu haben – dann steht der Speicherung im Tabellenfeld einer Datenbank nichts mehr im Wege.

Übrigens: Die wesentliche Codezeile, mit deren Hilfe aus der jeweiligen Zeichenkette eine gleichlautende Variable wird, ist `${$name} = $wert`.

1.4.9 Vordefinierte Variablen

PHP bietet jedem ausgeführten Skript eine Vielzahl von vordefinierten Variablen an. Wir werden uns im folgenden Abschnitt lediglich auf einige wesentliche Variablen beschränken.

Umgebungs-Variablen (Environment-Variablen)

Diese Variablen werden aus der Umgebung, in der PHP läuft, in den globalen Namensbereich von PHP importiert. Viele werden durch die jeweilige Shell, in der PHP läuft, unterstützt bzw. gebildet. Da es verschiedenste Systemumgebungen mit den unterschiedlichsten Shells gibt, ist es nicht möglich, eine abschließende Liste der definierten Umgebungs-Variablen aufzustellen. Lesen Sie deshalb in der Anleitung zu Ihrer Shell nach, um eine Liste dieser systembezogenen Variablen zu erhalten.

PHP-Variablen

Diese Variablen werden durch PHP selbst erzeugt. `$HTTP_*_VARS`-Variablen stehen nur zur Verfügung, wenn die Option *track_vars* in der *php.ini* auf »*on*« gesetzt ist. Wenn dies der Fall ist, werden diese Variablen immer gesetzt, selbst wenn es leere Arrays sind. Das verhindert, dass ein böswilliger Benutzer diese Variablen manipuliert.

Wenn `register_globals` aktiviert ist, stehen auch diese Variablen im globalen Namensbereich des Skripts zur Verfügung, z.B. getrennt von den Arrays `$HTTP_*_VARS` und `$_*`.

Variable	*Beschreibung*
$argv	Ein Array von Argumenten, die dem Skript übergeben werden. Wird das Skript an der Befehlszeile aufgerufen, ermöglicht dies C-ähnlichen Zugriff auf die Kommandozeilenparameter. Beim Aufruf per GET-Methode enthält dieses Array die Abfragewerte.
$argc	Anzahl der per Kommandozeile dem Skript übergebenen Parameter, wenn das Skript aus der Kommandozeile aufgerufen wurde.
$PHP_SELF	Der Dateiname des gerade ausgeführten Skripts, relativ zum Wurzelverzeichnis des Dokuments. Bei Kommandozeilenaufrufen ist diese Variable nicht verfügbar.
$HTTP_COOKIE_VARS	Ein assoziatives Array von Variablen, das dem aktuellen Skript über HTTP-Cookies übergeben wurde.
$_COOKIE	Ein assoziatives Array von Variablen, das dem aktuellen Skript über HTTP-Cookies übergeben wurde. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.
$HTTP_GET_VARS	Ein assoziatives Array von Variablen, das dem aktuellen Skript per HTTP-GET-Methode übergeben wurde.
$_GET	Ein assoziatives Array von Variablen, das dem aktuellen Skript per HTTP-GET-Methode übergeben wurde. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.

Variable	*Beschreibung*
$HTTP_POST_VARS	Ein assoziatives Array aus Variablen, welches dem aktuellen Skript per HTTP-POST-Methode übergeben wurde.
$_POST	Ein assoziatives Array aus Variablen, welches dem aktuellen Skript per HTTP-POST-Methode übergeben wurde. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.
$HTTP_POST_FILES	Ein assoziatives Array aus Variablen, das Informationen über per HTTP POST-Methode hochgeladene Dateien enthält.
$_FILES	Ein assoziatives Array aus Variablen, das Informationen über per HTTP POST-Methode hochgeladene Dateien enthält. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.
$HTTP_ENV_VARS	Ein assoziatives Array aus Variablen, die dem aktuellen Skript über die Umgebung zur Verfügung stehen.
$_ENV	Ein assoziatives Array aus Variablen, die dem aktuellen Skript über die Umgebung zur Verfügung stehen. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.
$HTTP_SERVER_VARS	Ein assoziatives Array aus Variablen, die dem aktuellen Skript vom jeweiligen HTTP-Server übermittelt werden.
$_SERVER	Ein assoziatives Array aus Variablen, die dem aktuellen Skript vom jeweiligen HTTP-Server übermittelt werden. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.
$HTTP_SESSION_VARS	Ein assoziatives Array aus Session-Variablen, die dem aktuellen Skript übergeben wurden.
$_SESSION	Ein assoziatives Array aus Session-Variablen, die dem aktuellen Skript übergeben wurden. Automatisch global in jedem Geltungsbereich. Werden dem Array $_SESSION neue Einträge hinzugefügt, werden diese automatisch als Session-Variablen registriert, so als ob die Funktion `session_register()` aufgerufen worden wäre. Eingeführt in PHP 4.1.0.
$_REQUEST	Ein assoziatives Array zusammengesetzt aus den GET-, POST- und Cookie-Variablen. Mit anderen Worten alle Informationen, die vom Benutzer kommen und denen aus Sichtweise der Sicherheit nicht zu trauen ist. Automatisch global in jedem Geltungsbereich. Eingeführt in PHP 4.1.0.

Hinweis: Die neuen »Superglobals« bzw. `register_globals` stehen seit der PHP-Version 4.1.0. zur Verfügung. Dies sind die Arrays `$_GET`, `$_POST`, `$_ENV`, `$_SERVER`, `$_COOKIE`, `$_REQUEST`, `$_FILES` und `$_SESSION`. Sie werden informell als Superglobals bezeichnet, da sie immer zur Verfügung stehen, ohne Berücksichtigung des Geltungsbereichs. Damit sind die `$HTTP_*_VARS`-Arrays veraltet. Im folgenden Abschnitt erfahren Sie mehr darüber.

1.4.10 Einsatz von register_globals

Welche PHP-Version verwenden Sie? Finden Sie es heraus! Prüfen Sie bei dieser Gelegenheit, wie die Variable *register_globals* bei Ihnen eingestellt ist. Dies gelingt mit Hilfe des PHP-Skripts

```
<?

phpinfo();

?>
```

Im Browserfenster wird eine umfangreiche Übersichtseite erscheinen. Scrollen Sie in dieser Datei ein Stück nach unten und überprüfen Sie die Einstellung der Variablen

`register_globals`.

Die Einstellung der Variablen können Sie in der Konfigurationsdatei *php.ini* jederzeit selbst vornehmen. Der Eintrag in der Konfigurationsdatei stellt sich wie folgt dar:

```
register_globals = On
```

Seit der PHP-Version 4.2 wird diese Variable jedoch standardmäßig auf *Off* gesetzt.

```
register_globals = Off
```

Sie merken es vor allem dann, wenn Sie Ihren eigenen Webserver betreiben und updaten. Schalten Sie daher bei Problemen mit Ihren PHP-Skripts (vorerst) zurück auf *On.* Nach einem Neustart des Webservers stehen die geänderten Einstellungen zur Verfügung. Die meisten Provider mit PHP-Unterstützung haben in der Regel noch nicht auf *register_globals = Off* umgeschaltet. Zu groß wäre wohl der Aufschrei vieler Kunden. Schließlich würde diese Maßnahme bedeuten, dass viele Skripts auf einen Schlag nicht mehr funktionierten. Vor allem bei umfangreichen Projekten wäre die Umstellung mit einem erheblichen Aufwand verbunden.

Was hat es mit der Änderung auf sich?

Hinter dieser Änderung stecken z.B. die Informationen aus »GET« und »POST«, »COOKIES« und »SERVER« – also praktisch der gesamte Bereich externer Variablenquellen. Und diese Werte stehen nach der Änderung der Einstellung für *register_globals* nun nicht mehr zur Verfügung.

Die Informationen aus dieser Quelle können nun nicht mehr so einfach per $Variablennamen ausgelesen werden. Am Beispiel einer Formularauswertung wollen wir Ihnen dies verdeutlichen. Bisher war PHP wirklich einfach gestrickt. Der Name eines Formularfelds wurde automatisch zur Variablen.

```
<input type="text" name="telefon">
```

Im auswertenden PHP-Skript haben Sie folgendermaßen auf den Inhalt des Formularfelds zugegriffen:

```
$telefon
```

Dabei spielte es übrigens keine Rolle, ob das Formular per

```
method = "post"
```

oder

```
methode = "get"
```

abgeschickt wurde. Selbst die Werte von Cookies konnte man anhand ihres Namens ermitteln. Diese Schreibweise stellt sich als recht bequem dar, aber auch äußerst problematisch, wenn es um Eindeutigkeit und Sicherheit geht.

Probleme der Schreibweise – Eindeutigkeit

Es kann durchaus zu Verwechslungen kommen, vor allem wenn Sie nicht zu den diszipliniertesten Entwicklern gehören. Sie erzeugen etwa ein Formularfeld, das *name* heißt. Zufälligerweise steckt auch ein gleichnamiges Cookie in Ihrem Quellcode. Dann haben Sie ein Problem, denn sowohl der Inhalt des Formularfelds als auch der Wert des Cookies stehen nun über `$name` zur Verfügung. Diese Schwierigkeiten könnte man mit einer konsequenten Variablenbennenung vermeiden. Beginnen Sie Cookie-Variablen grundsätzlich mit einem kleinen `c` und Formular-Variablen mit einem kleinen `f`.

Probleme der Schreibweise – Sicherheit

Kommen wir nun zum Sicherheitsproblem. Es gibt bei schlampig programmiertem Code viele Angriffsmöglichkeiten für potentielle Hacker. Schauen Sie sich einmal folgenden Teil eines PHP-Skripts an:

```
if ($pw=="g1882m") {
       $login = true;
}
```

Hier wird mit der Signalvariablen `$login` gearbeitet. Nur bei Kenntnis des richtigen Passworts soll sie auf `true` gesetzt werden. Dummerweise wurde diese Variable am Anfang des Beispiels nicht mit `false` initialisiert. Kein großes Problem, denkt man – bei dem recht komplexen Passwort – aber im Gegenteil: Auf diese Weise kann ein Angreifer nun ganz einfach ohne Kenntnis des Passworts in den geschützten Bereich gelangen. Wie? Er muss an den URL lediglich `?login=true` oder `?login=1` anhängen.

```
http://localhost/beispiel.php?login=true
```

Schon wird die Variable `$login` auf `true` gesetzt und der Zugang ist auch ohne Passwortkenntnis möglich. Dies wiederum stellt ein riesiges Sicherheitsloch dar.

Neue Schreibweise = mehr Eindeutigkeit und Sicherheit

Dieses Problem hat das PHP-Entwicklerteam erkannt und spätestens mit Einführung von PHP 4.1 elegant behoben. Sämtliche über `method = "post"` versendeten Formulardaten sind im neuen Array `$_POST` gespeichert, die per `method = "get"` erhältlichen Daten dagegen im neuen Array `$_GET`. Dazu gehören auch die an den URL angehängten Parameter. Es handelt sich bei `$_POST` und `$_GET` übrigens um assoziative Arrays. Der Schlüssel wird

aus dem Namen des entsprechenden Formularfelds bzw. der entsprechenden Cookie-Variablen gebildet. Wenn der Wert des URL-Anhangs `?login=true` erfasst werden soll, gelingt dies über `$_GET["login"]`. Bei konsequent abgeschalteten `register_globals` ist eine Verwechslung mit einer Variablen `$login` nun nicht mehr möglich.

Um die alten Sicherheitslücken vollkommen zu schließen, wird seit PHP-Version 4.2 daher auch `register_globals = Off` als Standard gesetzt. Diese Einstellung bietet optimale Sicherheit und ist daher für zukünftige Projekte dringend zu empfehlen.

Erweiterungen

Wir wollen Ihnen natürlich nicht vorenthalten, dass seit Version 4.1 neben `$_POST` und `$_GET` folgende weitere assoziative Arrays eingeführt wurden:

`$_COOKIE`

Dieses Array enthält sämtliche Cookie-Variablen. Mit `$_COOKIE["besucher"]` würden Sie den Wert des Cookies *besucher* ermitteln.

`$_REQUEST`

Dieses Array nimmt eine Sonderposition ein. Es enthält sämtliche Werte aus `$_POST`, `$_GET` und `$_COOKIE`. Wir empfehlen, den Einsatz von `$_REQUEST` zu vermeiden, da es die Bemühungen um mehr Eindeutigkeit zunichte macht. Mit `$_REQUEST["login"]` können Sie z.B. sowohl auf ein Formularfeld namens `login` als auch auf das gleichnamige Cookie zurückgreifen.

Mit den neuen Variablen entfallen übrigens die bis Version 4.1 gültigen Arrays:

- `$HTTP_POST_VARS`
- `$HTTP_GET_VARS`
- `$HTTP_COOKIE_VARS`

Dies bedeutet wiederum eine Vereinfachung der Schreibweise.

Zusätzlich werden die neuen Array-Variablen `$_SERVER`, `$_ENV` und `$_SESSION` zur Verfügung gestellt. `$_ENV` ist für die Umgebungsvariablen und `$_SESSION` für das Session-Management verantwortlich. Am interessantesten ist sicher die erstgenannte Variable `$_SERVER`. Sie enthält sämtliche Servervariablen, also die Variablen, die der Webserver übergibt.

Wenn man früher Pfad und Dateinamen ermitteln wollte, schrieb man `$PHP_SELF`. Heute wird via `$_SERVER["PHP_SELF"]` auf diesen Wert zugegriffen.

Tipps zur neuen Schreibweise

Auf den ersten Blick wird die Schreibweise durch die neue Array-Syntax komplizierter. So konnte man ein Formularfeld im PHP-Skript nach der alten Schreibweise unkompliziert auslesen und weiterverarbeiten.

```
<input type="text" name="telefon">
```

Auslesen und Verarbeiten:

```
echo "Sie haben folgende Nummer $telefon";
```

Eine Verkettung des umgebenden Strings mit der Variablen ist dabei nicht nötig. Nach der neuen Syntax geht das offenbar nicht mehr so einfach. Versuchen Sie es noch einmal:

```
echo "Sie haben folgende Nummer $_POST['telefon']";
```

Die Ausgabe enthält nicht den Inhalt der Variablen, sondern es wird der String `$_POST['telefon']` ausgegeben. Dabei nützt es auch nichts, dass Sie die Regeln beherzigen, bei der Verschachtelung von Anführungszeichen nur ungleiche Anführungszeichen zu verwenden. Es wurden für den Array-Schlüssel die einfachen Anführungszeichen und für den gesamten String die doppelten Anführungszeichen verwendet. Doch das hilft alles nichts. Es muss verkettet werden:

```
echo "Sie haben folgende Nummer" . $_POST['telefon'];
```

Vor allem bei umfangreichen Auswertungen wird es dadurch schnell unübersichtlich und kompliziert. Doch folgender Trick könnte Ihnen bei Ihrer Arbeit behilflich sein. Lassen Sie die normalerweise üblichen Anführungszeichen vor und hinter dem Array-Schlüssel einfach weg:

```
echo "Sie haben folgende Nummer $_POST[telefon]";
```

Diese anführungszeichenfreie Schreibweise ist immer dann möglich, wenn die Array-Variable selbst innerhalb eines Anführungszeichenpaares steht. In allen anderen Fällen sollten Sie den Schlüssel je nach Bedarf stets mit einfachen oder doppelten Anführungszeichen versehen.

1.5 Konstanten

Eine Konstante ist ein Bezeichner (Name) für eine simple Variable. Wie der Name schon ausdrückt, kann sich der Wert einer Konstanten zur Laufzeit eines Skripts nicht ändern. Eine Konstante unterscheidet zwischen Groß- und Kleinschreibung (case-sensitive). Nach gängiger Konvention werden Konstanten immer in Großbuchstaben geschrieben.

- Eine Konstante können Sie über die Funktion `define()` definieren. Einmal definiert, kann eine Konstane weder verändert noch gelöscht werden.
- Konstanten können nur skalare Daten wie boolean, integer, float und string enthalten.

Unterschiede zwischen Konstanten und Variablen:

- Konstanten haben kein Dollarzeichen ($) vorangestellt.
- Konstanten können nur über die Funktion `define()` definiert werden, nicht durch einfache Zuweisung.
- Konstanten können überall definiert werden, und auf Ihren Wert können Sie ohne Rücksicht auf Namensraumregeln von Variablen zugreifen.

- Sobald Konstanten definiert sind, können sie nicht neu definiert oder gelöscht werden.
- Konstanten können nur skalare Datenwerte besitzen.

```
<?php
define("SPRUCH", "Willkommen!");
// Ausgabe - "Willkommen!"
echo SPRUCH;
?>
```

Konstanten sind in allen Programmier- und Skriptsprachen nützlich, um feste, immer wieder benötigte Werte mit verständlichen Begriffen zu umschreiben. Der Einsatz erhöht die Lesbarkeit des Quellcodes.

1.5.1 Vordefinierte Konstanten

Viele dieser Konstanten werden jedoch von verschiedenen Erweiterungen definiert, die nur zur Verfügung stehen, wenn diese Erweiterungen selbst zur Verfügung stehen, entweder über dynamisches Laden zur Laufzeit oder weil sie einkompiliert sind. Eine Auswahl von Konstanten haben wir für Sie in der folgenden Tabelle zusammengestellt:

Konstante	*Beschreibung*
__LINE__	Liefert die aktuelle Zeilennummer einer Datei. Wird die Konstante in einer Datei verwendet, die per `include()` oder `require()` eingebunden wurde, liefert sie die Zeilennummer innerhalb der eingebundenen Datei.
__FILE__	Liefert den vollständigen Pfad- und Dateiname einer Datei. Wird diese Konstante in einer Datei verwendet, die per `include()` oder `require()` eingebunden wurde, liefert sie den Pfad- und Dateiname der eingebundenen Datei, nicht den der aufrufenden Datei.
__FUNCTION__	Der Name einer Funktion. Steht seit PHP 4.3.0 zur Verfügung.
__CLASS__	Der Name einer Klasse. Steht seit PHP 4.3.0 zur Verfügung.
__METHOD__	Der Name einer Klassenmethode. Steht seit PHP 5.0 zur Verfügung.
NULL	Der Wert NULL. NULL bedeutet im Gegensatz zu einem leeren String oder der Zahl 0, dass keine Eingabe erfolgt ist. Dies ist beispielsweise bei der Abfrage von Datenbankfeldern von Bedeutung. Die Prüfung des Datentyps einer Variablen, die NULL enthält, mit der Funktion `gettype()`, ergibt NULL.
PHP_OS	Der Name des Betriebssystems, auf dem der PHP-Interpreter ausgeführt wird.
PHP_VERSION	Eine Zeichenkette, die die Versionsnummer des PHP-Interpreters enthält.
TRUE	Der Wert `Wahr (1)`. Die Konstante TRUE existiert seit PHP 4.0.
FALSE	Der Wert `Falsch (0)`. Die Konstante FALSE existiert seit PHP 4.0.
E_ERROR	Fehler, der sich von einem *parsing error* unterscheidet. Die Ausführung des Skripts wird beendet.

Konstante	*Beschreibung*
E_WARNING	Warnung, das aktuelle Skript wird jedoch weiter ausgeführt.
E_PARSE	Ungültige Syntax in der Skriptdatei. Die Ausführung des Skripts wird beendet.
E_NOTICE	Anmerkung. Hinweis auf mögliche Fehler. Das aktuelle Skript wird jedoch weiter ausgeführt.
E_CORE_ERROR	Fehler, welcher während der Initialisierung des PHP-Interpreters auftritt. Die Ausführung des Skripts wird beendet.
E_CORE_WARNING	Warnung, welche während der Initialisierung des PHP-Interpreters auftritt. Das aktuelle Skript wird jedoch weiter ausgeführt.
E_STRICT	Wurde zur Abwärtskompatibilität seit PHP 5.0 eingeführt.

1.6 Operatoren

Bevor Sie die in PHP zur Verfügung stehenden Operatoren kennen lernen, gibt es eine kurze Einführung der Begriffe Vorrang (Priorität) und Assoziativität der Operatoren.

1.6.1 Operator-Rangfolge

Die Operator-Rangfolge legt fest, wie »eng« ein Operator zwei Ausdrücke miteinander verbindet. Zum Beispiel ist das Ergebnis des Ausdrucks 1 + 5 * 3 *16* und nicht *18*, da der Multiplikationsoperator *(*)* in der Rangfolge höher steht als der Additionsoperator (+). Wenn nötig, können Sie Klammern setzen, um die Rangfolge der Operatoren zu beeinflussen. Zum Beispiel: (1 + 5) * 3 ergibt 18.

In dieser Tabelle sind alle PHP-Operatoren und ihre Assoziativität vom höchsten bis zum niedrigsten Vorrang aufgeführt.

Operator	*Beschreibung*	*Assoziativität*
Höchster Vorrang		
new	Objekt zuweisen	keine Richtung
[ ]	Array-Element	rechts
+	unäres Plus	rechts
-	unäres Minus	rechts
~	Bit-Komplement	rechts
!	logisches NOT	rechts
++	Post-Inkrement	rechts
--	Post-Dekrement	rechts
( )	Funktionsaufruf	rechts
++	Prä-Inkrement	rechts
--	Prä-Dekrement	rechts

Operator	*Beschreibung*	*Assoziativität*
*	*	links
/	/	links
%	Modulo	links
.	Strukturelement	links
+	+	links
-	-	links
<<	bitweise Verschiebung nach links	links
>>	bitweise Verschiebung nach rechts	links
>>>	bitweise Verschiebung nach rechts (ohne Vorzeichen)	links
<	kleiner als	keine Richtung
<=	kleiner als oder gleich	keine Richtung
>	größer als	keine Richtung
>=	größer als oder gleich	keine Richtung
==	gleich	keine Richtung
!=	ungleich	keine Richtung
===	strikt gleich	keine Richtung
&	bitweises AND	links
^	bitweises XOR	links
|	bitweises OR	links
&&	logisches AND	links
||	logisches OR	links
?:	bedingt	links
=	Zuweisung	links
*=, /=, %=, +=, -=, &=, |=, ^=, ~=, <<=, >>=, >>>=	zusammengesetzte Zuweisung	links
,	mehrfache Auswertung	links
Niedrigster Vorrang		

1.6.2 Vorrang der Operatoren

In der Operatorliste wird der Begriff »Höchster/Niedrigster Vorrang« verwendet. Der Vorrang der Operatoren bestimmt, in welcher Reihenfolge die Operationen ausgeführt werden. Operatoren mit höherem Vorrang werden vor denen mit einem niedrigeren Vorrang ausgeführt.

Beispiel:

```
$summe = 10 + 5 * 2;                    // Ergebnis: 20
```

Der Multiplikationsoperator (*) hat einen höheren Vorrang als der Additionsoperator (+), deswegen wird die Multiplikation vor der Addition ausgeführt, »Punktrechung vor Strichrechnung«. Zudem hat der Zuweisungsoperator (=) den niedrigsten Vorrang, deswegen wird die Zuweisung erst ausgeführt, wenn alle Operationen auf der rechten

Seite abgeschlossen sind. Der Vorrang von Operatoren kann durch Verwendung von Klammern außer Kraft gesetzt werden. Um in dem obigen Beispiel die Addition zuerst auszuführen, müssten wir also schreiben:

```
$summe = (10 + 5) * 2;          // Ergebnis: 30
```

Wenn Sie sich in der Praxis einmal unsicher sind, welcher der von Ihnen verwendeten Operatoren den Vorrang besitzt, ist es äußerst sinnvoll, Klammern zu verwenden, um so die Berechnungsreihenfolge explizit vorzugeben.

Tipp: Durch Klammerung mit runden Klammern ( ) kann die vorgegebene Hierarchie überwunden werden.

1.6.3 Assoziativität der Operatoren

Wenn ein Ausdruck mehrere Operatoren enthält, die hinsichtlich ihrer Rangfolge gleichwertig sind, wird die Reihenfolge ihrer Ausführung durch ihre Assoziativität bestimmt. Dabei wird zwischen zwei Richtungen unterschieden.

- Linksassoziativität (in Links-Rechts-Richtung)
- Rechtsassoziativität (in Rechts-Links-Richtung)

Der Multiplikationsoperator ist beispielsweise linksassoziativ. Daher sind die beiden folgenden Anweisungen austauschbar.

Beispiel:

```
$summe = 5 * 10 * 2;            // Ergebnis: 100
$summe = (5 * 10) * 2;          // Ergebnis: 100
```

Eine tabellarische Übersicht über die Operatoren und ihre Assoziativität finden Sie in der Operatorliste am Ende dieses Abschnitts.

Nachdem die Zusammenhänge von Operatorvorrang und Assoziativität geklärt wurden, werden Sie jetzt die Operatoren kennen lernen.

1.6.4 Arithmetische Operatoren

Addition

Der Additionsoperator (+) addiert die beiden numerischen Operanden.

Beispiel:

```
// Addition
$summe = 7 + 3;                 // Ergebnis: 10
$summe = 13 + 9 + 1;            // Ergebnis: 23
```

Subtraktion

Der Subtraktionsoperator (-) subtrahiert seinen zweiten Operanden vom ersten. Beide Operanden müssen Zahlen sein.

Beispiel:

```
// Subtraktion
$summe = 10 - 5;              // Ergebnis: 5
$summe = 1.5 - 0.5;           // Ergebnis: 1
```

Wenn (-) als unärer Operator vor einem einzigen Operanden eingesetzt wird, führt er eine unäre Negation durch, d.h., eine positive Zahl wird in die entsprechende negative umgewandelt, und umgekehrt.

Beispiel:

```
// Unäre Negation
$summe = -5;                  // Ergebnis: -5
$summe = - (+5);              // Ergebnis: -5
$summe = - (-5);              // Ergbenis: 5
// Nicht so!
$summe = --5;                 // Ergebnis: 4
```

Multiplikation

Der Multiplikationsoperator (*) multipliziert seine beiden Operanden, auch hier müssen beide Operanden Zahlen sein.

Beispiel:

```
// Multiplikation
$summe = 10 * 2;              // Ergebnis: 20
$summe = 5.75 * 2;            // Ergebnis: 11.5
```

Division

Der Divisionsoperator (/) dividiert seinen ersten Operanden durch den zweiten. Beide Operanden müssen Zahlen sein.

Beispiel:

```
// Division
$summe = 10 / 2;              // Ergebnis: 5
$summe = 5.75 / 2;            // Ergebnis: 2.875
```

Achtung: Der Divisions-Operator ("/") gibt immer eine Fließkommazahl zurück, sogar wenn die zwei Operanden Ganzzahlen sind (oder Zeichenketten, die nach Ganzzahlen umgewandelt wurden).

Modulo

Der Modulo-Operator (%) bildet den Rest aus einer Division zweier Operanden. Beide Operanden müssen Zahlen sein. Modulo ist also nichts anderes als die Ganzahldivision mit Rest. Dabei bildet der Rest der Division das Ergebnis der Modulo-Operation.

Beispiel:

```
// Modulo
$summe = 10 % 2;              // Ergebnis: (10 / 2 = 5) Rest 0
$summe = 10 % 3;              // Ergebnis: (10 / 3 = 3) Rest 1
```

Operator	*Bezeichnung*	*Bedeutung*
+	Positives Vorzeichen	+$a ist/entspricht $a.
-	Negatives Vorzeichen	-$a kehrt das Vorzeichen um.
+	Addition	$a + $b ergibt die Summe von $a und $b.
-	Subtraktion	$a – $b ergibt die Differenz von $a und $b.
*	Multiplikation	$a * $b ergibt das Produkt aus $a und $b.
/	Division	$a / $b ergibt den Quotienten von $a und $b.
%	Restwert (Modulo)	$a % $b ergibt den Restwert der Division von $a durch $b.

1.6.5 Zuweisungsoperator

Wie Sie bereits im Abschnitt zu den Variablen gesehen haben, wird in PHP (=) verwendet, um einer Variablen einen Wert zuzuweisen.

Beispiel:

```
$vorname = "Matze";
```

Auch wenn man eine solche PHP-Zeile nicht als Ausdruck ansieht, der ausgewertet werden kann und einen Wert hat, handelt es sich doch wirklich um einen Ausdruck. Technisch gesehen ist (=) ein Operator. Der Operator (=) erwartet als linken Operanden eine Variable. Als rechter Operand wird ein beliebiger Wert eines beliebigen Typs erwartet. Der Wert eines Zuweisungsausdrucks ist der Wert des rechten Operanden. Da (=) als Operator definiert ist, kann er auch als Bestandteil komplexerer Ausdrücke verwendet werden.

Beispiel:

```
$zahlEins = 200;
$zahlZwei = 250;
$pruefen = zahlEins == zahlZwei;      // Ergebnis: false
```

Hinweis: Wenn Sie so etwas verwenden wollen, sollte Ihnen vorher der Unterschied zwischen den Operatoren (=) und (==) vollkommen klar sein.

Der Zuweisungsoperator ist von rechts nach links assoziativ, das bedeutet, dass bei mehreren Zuweisungsoperatoren innerhalb eines einzigen Ausdrucks von rechts nach links

ausgewertet wird. Als Folge davon kann man Code wie den folgenden schreiben, um mehreren Variablen jeweils denselben Wert zuzuweisen.

Beispiel:

```
// Initialisierung mehrerer Variablen in einem Ausdruck
$i = $j = $k = 100;
```

Zuweisung mit Operation

Neben dem Zuweisungsoperator (=) unterstützt PHP noch eine Reihe weiterer Zuweisungsoperatoren, die eine Kurzform bzw. Kurznotation dafür darstellen, dass eine Zuweisung mit einer anderen Operation verbunden wird.

Beispiel:

```
// Initialisierung
$preis = 10.00;
$mwst = 1.60;
// Kurzform
$preis +=  $mwst;                    // Ergebnis: 11.6
// Gleichbedeutend
$preis = $preis + $mwst              // Ergebnis: 11.6
```

Entsprechend gibt es auch -=, *=, /=, %= usw. Die nachfolgende Tabelle führt all diese Operatoren auf.

Beispiel:

```
// Initialisierung
$zahlEins = 50;
$zahlZwei = 25;

// Kurzformen
$zahlEins += $zahlZwei;              // Ergebnis: 75
$zahlEins -= 5;                      // Ergebnis: 70
$zahlEins *= 2;                      // Ergebnis: 140
$zahlZwei /= 5;                      // Ergebnis: 5
$zahlZwei %= 2;                      // Ergebnis: 1
```

Abschließend eine Übersicht über die Zuweisungsoperatoren in PHP.

Operator	*Bezeichnung*	*Bedeutung*
=	Einfache Zuweisung	$a = $b weist $a den Wert von $b zu und liefert $b als Rückgabewert.
+=	Additionszuweisung	$a += $b weist $a den Wert von $a + $b zu und liefert $a + $b als Rückgabewert.
-=	Subtraktionszuweisung	$a -= $b weist $a den Wert von $a – $b zu und liefert $a – $b als Rückgabewert.
*=	Multiplikationszuweisung	$a *= $b weist $a den Wert von $a * $b zu und liefert $a * $b als Rückgabewert.

Operator	Bezeichnung	Bedeutung
%=	Modulozuweisung	$a %= $b weist $a den Wert von $a % $b zu und liefert $a % $b als Rückgabewert.
/=	Divisionszuweisung	$a /= $b weist $a den Wert von $a / $b zu und liefert $a / $b als Rückgabewert.
.=	Zeichenkettenzuweisung	$a .= $b weist $a den Wert von $a . $b zu und liefert $a . $b als Rückgabewert.

Zuweisung »by reference«

Im Zusammenhang mit Zuweisungen ist noch ein wichtiger Punkt zu beachten:

Bei den bisher betrachteten Zuweisungen wird der Wert einer Variablen einer anderen Variablen zugewiesen. Es existieren also zwei unterschiedliche Variablen und somit auch zwei unterschiedliche Speicherbereiche, die nach der erfolgten Zuweisung zwar denselben Wert aufweisen, aber ansonsten völlig unabhängig voneinander im Arbeitsspeicher existieren. Diese Zuweisungsart wird auch als Zuweisung »by value« bezeichnet.

Seit PHP 4 steht Ihnen jedoch noch eine weitere Form der Zuweisung zur Verfügung, bei der nach erfolgter Zuweisung beide Variablen auf denselben Speicherbereich verweisen. In diesem Zusammenhang spricht man auch von Zuweisungen »by reference«. Wenn Sie Zuweisungen »by reference« vornehmen, weisen Sie also nicht einer Variablen den Wert einer anderen zu, sondern einer Variablen den Speicherbereich einer anderen, so dass im Ergebnis jetzt beide Variablen auf denselben Speicherbereich verweisen und somit nur noch eine Variable existiert, die allerdings zwei unterschiedliche Variablennamen besitzt.

Beispiel:

```
<?
$vorname = "Matthias";
$meinname = &$vorname;
// Ausgabe - Matthias
echo $meinname;
// Ausgabe - Matthias
echo $vorname;
$vorname = "Caroline";
// Ausgabe - Caroline
echo $meinname;
// Ausgabe - Caroline
echo $vorname;
$meinname = "Gülten";
// Ausgabe - Gülten
echo $meinname;
// Ausgabe - Gülten
echo $vorname;
?>
```

Operator	*Bezeichnung*	*Bedeutung*
$a = &$b	Zuweisung »by reference«	Der Speicherbereich der Variablen $a wird auf den Speicherbereich der Variablen $b gesetzt.

1.6.6 Vergleichsoperatoren

In diesem Abschnitt lernen Sie die Vergleichsoperatoren von PHP kennen. Es handelt sich hier um Operatoren, die Werte verschiedener Typen vergleichen und einen booleschen Wert (`true` oder `false`) liefern, je nach Ergebnis des Vergleichs. Die Vergleichsoperatoren werden am häufigsten in Konstruktionen wie `If`-Anweisungen und For/While-Schleifen eingesetzt. Hier haben sie die Aufgabe, den Programmablauf zu steuern.

Kleiner als

Der Operator (<) hat das Ergebnis `true`, wenn sein erster Operand kleiner ist als der zweite, sonst liefert er `false`. Die Operanden müssen Zahlen oder Strings sein. Strings werden dabei alphabetisch auf der Basis der Codewerte der Zeichen verglichen.

Beispiel:

```
// Kleiner als (mit Zahlen)
$preisHose = 75.50;
$preisJacke = 110.95;
$pruefen = $preisHose < $preisJacke;          // Ergebnis: true
```

Beispiel:

```
// Kleiner als (mit Strings)
$kundeEins = "Fred";
$kundeZwei = "Toni";
$pruefen = $kundeEins < $kundeZwei;           // Ergebnis: true
```

Größe als

Der Operator (>) hat das Ergebnis `true`, wenn sein erster Operand größer ist als der zweite, sonst liefert er `false`. Die Operanden müssen Zahlen oder Strings sein. Auch hier werden die Strings alphabetisch auf der Basis der Codewerte der Zeichen verglichen.

Beispiel:

```
// Grösser als (mit Zahlen)
$preisBrille = 65;
$preisUrlaub = 1150;
$pruefen = $preisUrlaub > $preisBrille;       // Ergebnis: true
```

Beispiel:

```
// Grösser als (mit Strings)
$kundeEins = "Timo";
$kundeZwei = "Bernd";
$pruefen = $kundeEins > $kundeZwei;          // Ergebnis: true
```

Kleiner oder gleich

Der Operator (<=) hat das Ergebnis `true`, wenn sein erster Operand kleiner als der zweite oder gleich diesem ist, sonst liefert er `false`. Die Operanden müssen Zahlen oder Strings sein, und Strings werden dabei alphabetisch auf der Basis der Codewerte der Zeichen verglichen.

Beispiel:

```
// Kleiner oder gleich (mit Zahlen)
$preisBrille = 65;
$preisUrlaub = 1150;
$pruefen = $preisBrille <= $preisUrlaub;     // Ergebnis: true

$preisBürste = 5.95;
$preisEimer = 5.95;
$pruefen = $preisBürste <= $preisEimer;      // Ergebnis: true
```

Beispiel:

```
// Kleiner oder gleich (mit Strings)
$kundeEins = "Bernd";
$kundeZwei = "Timo";
$pruefen = $kundeEins <= $kundeZwei;         // Ergebnis: true

$wortEins = "Sonntag";
$wortZwei = "Sonntag";
$pruefen = $wortEins <= $wortZwei;           // Ergebnis: true
```

Größer oder gleich

Der Operator (>=) hat das Ergebnis `true`, wenn sein erster Operand größer als der zweite oder gleich diesem ist, sonst liefert er `false`. Die Operanden müssen Zahlen oder Strings sein, und auch hier werden Strings alphabetisch auf der Basis der Codewerte der Zeichen verglichen.

Beispiel:

```
// Grösser oder gleich (mit Zahlen)
$preisAuto = 35000;
$preisUrlaub = 1150;
$pruefen = $preisAuto >= $preisUrlaub;       // Ergebnis: true

$preisBürste = 5.95;
$preisEimer = 5.95;
$pruefen = $preisBürste >= $preisEimer;      // Ergebnis: true
```

Beispiel:

```
// Grösser oder gleich (mit Strings)
$kundeEins = "Thomas";
$kundeZwei = "Caroline";
$pruefen = $kundeEins >= $kundeZwei;          // Ergebnis: true

$wortEins = "Sonntag";
$wortZwei = "Sonntag";
$pruefen = $wortEins >= $wortZwei;            // Ergebnis: true
```

Hier noch ein Beispiel mit jeweils einer Kontrollstruktur – eine `If`-Anweisung und eine `For`-Schleife.

Beispiel:

```
// Initialisierung
$preisAuto = 27500;
$preisBoot = 22500;

// Nach dem Vergleich enthält die Variable kaufen "Nein!"
if ($preisAuto <= $preisBoot) {
       $kaufen = "Ja!";
} else {
       $kaufen = "Nein!";
}
```

Beispiel:

```
// for-Schleife
// Ergebnis im Ausgabefenster 0 1 2 3 4 5 6 7 8 9 10
for ($i=0;$i<=10;$i++) {
       echo $i;
}
```

Achtung: Die Vergleichsoperatoren vergleichen zwei Strings in Bezug auf deren Anordnung zueinander. Der Vergleich benutzt dabei die alphabetische Ordnung. Zu beachten ist, dass diese Ordnung auf der von PHP verwendeten Zeichenkodierung Latin-1 (ISO8859-1) beruht, die eine Erweiterung des ASCII-Zeichensatzes darstellt. In dieser Kodierung kommen alle Großbuchstaben (ohne Umlaute) vor sämtliche Kleinbuchstaben, d.h., die Großbuchstaben sind kleiner!

Beispiel:

```
// Gross- u. Kleinbuchstaben Vergleich
$ortEins = "Zoo";
$ortZwei = "spielplatz";
$pruefen = $ortEins < $ortZwei;               // Ergebnis: true
```

Abschließend eine Übersicht über die Vergleichsoperatoren in PHP.

Operator	*Bezeichnung*	*Bedeutung*
<	Kleiner als	$a < $b ergibt true, wenn $a kleiner $b ist.
>	Größer als	$a > $b ergibt true, wenn $a größer $b ist.
<=	Kleiner oder gleich	$a <= $b ergibt true, wenn $a kleiner oder gleich $b ist.
>=	Größer oder gleich	$a >= $b ergibt true, wenn $a größer oder gleich $b ist.

1.6.7 Gleichheitsoperatoren

Gleichheit

Der Operator (==) liefert `true`, wenn seine beiden Operanden gleich sind; sind sie ungleich, liefert er `false`. Die Operanden können beliebige Typen haben, aber die Definition von gleich hängt vom Typ ab.

Noch etwas sollten Sie berücksichtigen: Normalerweise sind zwei Variablen nicht gleich, wenn sie verschiedene Typen haben. Da PHP aber bei Bedarf automatisch Datentypen umwandelt, ist dies nicht immer zutreffend.

Beispiel:

```
echo $pruefen = "1" == 1;                    // Ergebnis: true
echo $pruefen = true == 1;                   // Ergebnis: true
echo $pruefen = false == 0;                  // Ergebnis: true
```

Achtung: Beachten Sie, dass der Gleichheitsoperator (==) etwas ganz anderes als der Zuweisungsoperator (=) ist, auch wenn Sie im Deutschen beide oft einfach als »gleich« lesen. Es ist wichtig, diese beiden Operatoren zu unterscheiden und in jeder Situation jeweils den richtigen Operator zu verwenden.

Ungleichheit

Auch hier gilt, der Operator (!=) liefert `true`, wenn seine beiden Operanden ungleich sind; sind sie gleich, liefert er `false`. Die Operanden können beliebige Typen haben, wie beim Operator (==).

Beispiel:

```
// Ungleichheit
$zahlEins = 999;
$zahlZwei = 99;
$pruefen = $zahlEins != $zahlZwei;           // Ergebnis: true
```

Beispiel:

```
// Initialisierung
$kundeEins = "Martin Klein";
$kundeZwei = "Fred Mustermann";
```

```
// Ergebnis: "Nicht identischer Kunde"
if ($kundeEins != $kundeZwei) {
        resultat = "Nicht identischer Kunde";
} else {
        resultat = "Identischer Kunde";
}
```

Strikte Gleichheit

Der Operator (===) funktioniert ähnlich wie der Gleichheitsoperator, führt jedoch keine Typumwandlung durch. Wenn zwei Operanden nicht denselben Typ aufweisen, gibt der strikte Gleichheitsoperator den Wert `false` zurück.

Beispiel:

```
// Initialiserung
$preisBuchEins = 45.95;
$preisBuchZwei = "45.95";

// Ergebnis: "Ungleich"
if ($preisBuchEins === $preisBuchZwei) {
        $pruefen = "Gleich";
} else {
        $pruefen = "Ungleich";
}
```

Strikte Ungleichheit

Der Operator (!==) funktioniert genau umgekehrt wie der strikte Gleichheitsoperator.

Beispiel:

```
// Initialiserung
$preisBuchEins = 45.95;
$preisBuchZwei = "45.95";

// Ergebnis: "Ungleich"
if ($preisBuchEins !== $preisBuchZwei) {
        $pruefen = "Ungleich";
} else {
        $pruefen = "Gleich";
}
```

Abeschließend eine Übersicht über die Gleichheitsoperatoren in PHP.

Operator	*Bezeichnung*	*Bedeutung*
==	Gleichheit	$a==$b ergibt true, wenn $a gleich $b ist.
===	Strikte Gleichheit	$a===$b ergibt true, wenn $a gleich $b ist vom gleichen Typ.
!=	Ungleichheit	$a!=$b ergibt true, wenn $a ungleich $b ist.
!==	Strikte Ungleichheit	$a!==$b ergibt true, wenn $a ungleich $b ist vom gleichen Typ.

1.6.8 Logische Operatoren

Logische Operatoren dienen zum Vergleichen boolescher Werte (`true` und `false`) und geben einen dritten booleschen Wert zurück. Bei der Programmierung werden sie normalerweise zusammen mit Vergleichsoperatoren verwendet, um auf diese Weise komplexe Vergleiche auszudrücken, die sich auf mehr als eine Variable beziehen.

Logisches Und

Der Operator (&&) ergibt dann und nur dann `true`, wenn gleichzeitig der erste und der zweite Operand `true` sind. Ergibt schon der erste Operand ein `false`, ist das Ergebnis ebenfalls `false`. Das ist der Grund dafür, weshalb sich der Operator (&&) gar nicht erst damit aufhält, den zweiten Operanden noch zu überprüfen.

Beispiel:

```
$wertEins = true;
$wertZwei = true;
// Beide Ausdrücke sind gleichwertig
// Ergebnis: true
echo $resultat = $wertEins && $wertZwei;
// Ergebnis: true
if ($wertEins && $wertZwei) $resultat = true;
```

Beispiel:

```
$wertEins = (10 * 2);
$wertZwei = (10 + 10);
// Beide Ausdrücke sind gleichwertig
// Ergebnis: true
if ($wertEins && $wertZwei) $resultat = true;
```

Um den logischen Operator (&&) noch besser zu verstehen, hier eine Wahrheitstabelle.

Operand 1	*Operand 2*	*Operand 1 && Operand 2*
true	false	false
false	true	false
true	true	true
false	false	false

Logisches Oder

Der Operator (||) ergibt nur dann `true`, wenn der erste oder der zweite Operand wahr ist oder auch beide gleichzeitig. Genau wie (&&) wertet auch dieser Operator den zweiten Operanden nicht aus, wenn der erste Operand das Ergebnis schon endgültig festlegt. Ergibt der erste Operand `true`, dann ist das Ergebnis ebenfalls `true`, der zweite Operand kann das Ergebnis nicht mehr ändern und wird daher nicht ausgewertet.

Beispiel:

```
$wertEins = (10 * 2);
$wertZwei = (10 + 10);
// Logische Operator (||) - OR
// Ergebnis: true
if ($wertEins || $wertZwei) $resultat = true;
```

Um den logischen Operator (||) noch besser zu verstehen, hier eine Wahrheitstabelle.

Operand 1	*Operand 2*	*Operand 1* || *Operand 2*
true	false	true
false	true	true
true	true	true
false	false	false

Logisches Nicht

Der Operator (!) ist ein unärer Operator, der vor seinem einzigen Operanden steht. Sein Zweck besteht darin, den booleschen Wert seines Operanden umzukehren.

Beispiel:

```
$wertEins = true;
// Logische Operator (!) - NICHT
// Ergebnis: false
$resultat = !$wertEins;
```

Um den logischen Operator (!) noch besser zu verstehen, hier eine Wahrheitstabelle.

Operand 1	*! Operand 1*
true	false
false	true

Zusammenfassend eine Übersicht über die logischen Operatoren in PHP.

Operator	*Bezeichnung*	*Bedeutung*
&& / and	Logisches UND (AND) Verknüpfung	$a && $b ergibt true, wenn sowohl $a als auch $b wahr sind. Ist $a bereits falsch, so wird false zurückgegeben und $b nicht mehr ausgewertet.
|| / or	Logisches ODER (OR) Disjunktion	$a || $b ergibt true, wenn mindestens einer der beiden Ausdrücke $a oder $b wahr ist. Ist bereits $a wahr, so wird true zurückgegeben und $b nicht mehr ausgewertet.
xor	Exklusiv-ODER (XOR)	$a xor $b ergibt true, wenn genau einer der beiden Ausdrücke $a oder $b wahr ist.
!	Logisches NICHT Negation	!$a ergibt false, wenn $a wahr ist, und true, wenn $a false ist.

1.6.9 Bit-Operatoren

Bitweise Operatoren wandeln Fließkommazahlen intern in 32-Bit-Ganzzahlen um. Welche Rechenoperation jeweils ausgeführt wird, hängt vom verwendeten Operator ab. In jedem Falle werden bei der Berechnung des Ergebniswertes jedoch die einzelnen Binärziffern (Bits) der 32-Bit-Ganzzahl unabhängig voneinander ausgewertet. Glücklicherweise werden Sie diese doch recht komplizierten Operatoren relativ selten bis gar nicht benötigen.

Wir wollen Ihnen jedoch die Vorzüge der Bitwise-Operatoren nicht vorenthalten. Die Bitwise-Operatoren werden von den meisten PHP-Entwicklern, wie bereits erwähnt, ignoriert, da sie es nicht gewohnt sind, binär zu arbeiten. Das Zahlensystem, welches nur zwei Werte kennt, nämlich 0 oder 1, ist einer Vielzahl von Entwicklern suspekt. Wir empfehlen Ihnen jedoch, den Bitwise-Operatoren eine Chance zu geben.

Fallbeispiel: Rechner-Tuning:

Stellen Sie sich vor, Sie betreiben einen Shop. Darin bieten Sie das Tunen von Rechnern an (Aufrüstung). Folgende Komponenten können nachgerüstet werden:

- Zweite Festplatte
- Netzwerkkarte
- DVD-Brenner
- TV-Karte

Nun könnten Sie zur Verwaltung dieser Komponenten pro Kunde Variablen verwenden, die die Wünsche des Kunden berücksichtigen.

```
$extraHD = true;
$netzkarte = true;
$brenner = true;
$tvkarte = true;
```

Einen Nachteil hat dieser Ansatz: Wir benötigen für jede Komponente eine separate Variable, die jeweils den booleschen Wert:

- true (installieren)
- false (nicht installieren)

speichert. Dies bedeutet natürlich auch, dass jede Variable Speicherplatz in Anspruch nimmt. Genau hierfür eignet sich hervorragend der Einsatz von Bitwise-Operatoren. Sie ermöglichen die Verwaltung von Daten auf binärer Ebene und lassen sich hervorragend kombinieren, so das Sie nicht mehr vier Variablen benötigen, sondern lediglich eine – Sie haben richtig gelesen, eine!

Hinweis: Dies schont den Speicher und ist vor allem um einiges schneller in der Verarbeitung, da Sie auf der untersten Ebene mit Ihrem Computer kommunizieren, sozusagen auf Maschinencode-Ebene, denn die binäre Ebene kennt nur 0/1.

Bit-Programmierung in die Praxis umzusetzen

Sie sollten sich zuerst im Klaren darüber sein, wie Ihr Computer Prozesse verarbeitet. In PHP gibt es die Möglichkeit, über die Bitwise-Operatoren diese Ebene zu ereichen und zu nutzen.

Ein binärer Zahlenwert wird in Zahlensequenzen von Nullen und Einsen gespeichert. Die Basis im binären Zahlensystem liegt bei 2. Um dieses Zahlensystem an unser Dezimalzahlensystem (mit Basis 10) anzupassen, müssen Sie die Beziehung zwischen beiden Systemen kennen. Hier einige binäre Zahlensequenzen, welche in Dezimalsequenzen umgewandelt werden, auf der linken Seite binär und auf der rechten dezimal (inkl. Umrechnung):

1 Bit 1 // 1: (1 x 1) = 1
2 Bit 10 // 2: (1 x 2) + (0 x 1) = 2
2 Bit 11 // 3: (1 x 2) + (1 x 1) = 3
3 Bit 100 // 4: (1 x 4) + (0 x 2) + (0 x 1) = 4
4 Bit 1000 // 8: (1 x 8) + (0 x 4) + (0 x 2) + (0 x 1) = 8
4 Bit 1001 // 9: (1 x 8) + (0 x 4) + (0 x 2) + (1 x 1) = 9
4 Bit 1100 // 12: (1 x 8) + (1 x 4) + (0 x 2) + (0 x 1) = 12
4 Bit 1111 // 15: (1 x 8) + (1 x 4) + (1 x 2) + (1 x 1) = 15

6 Bit 111111 // 63: (1 x 32) + (1 x 16) + (1 x 8) + (1 x 4) + (1 x 2) + (1 x 1) = 16

Wie Sie sehen, ist die Beziehung recht einfach, wenn man sich einmal die Umrechnung vor Augen hält. Immer wenn der binären Zahlensequenz eine weitere Ziffer hinzugefügt wird, wird sie die nächste Bit-Stufe erreichen und die Potenz wächst um das Doppelte.

Wenn Sie sich nun auf unser aktuelles Problem beziehen, haben wir es mit einer 4-Bit-Stufe zu tun. Da jede Variable durch ein Bit repräsentiert werden kann, würde sich diese Stufe hervorragend eignen.

Die vier Variablen lassen sich in einer einzigen Variable vereinen:

```
$auswahl = 1      // 0001; (extraHD)
$auswahl = 2      // 0010; (netzkarte)
$auswahl = 4      // 0100; (brenner)
$auswahl = 8      // 1000; (tvkarte)
```

Natürlich können Sie auch Komponenten miteinander kombinieren:

```
$auswahl = 5   // 0101; (extraHD und brenner)
$auswahl = 6   // 0110; (netzkarte und brenner)
$auswahl = 10  // 1010; (netzkarte und tvkarte)
$auswahl = 15  // 1111; (alle Komponenten)
```

Wie Sie sehen, ist die Zusammensetzung individuell zu bestimmen, ohne Probleme lassen sich diese Kombinationen in Form von Zahlen sichern. Das binäre System dahinter sorgt für die korrekte Zuordnung.

Wie können Sie nun die Komponenten hinzufügen, d.h., wie ereichen Sie es, den Brenner oder die TV-Karte der Variablen `$auswahl` zu übergeben, ohne die davor ausgewählte Komponente zu löschen?

Ein Beispiel:

```
$auswahl = 0; // Rechner ohne zusätzliche Komponenten
```

Hinzufügen von Komponenten – dabei können die einzelnen Komponenten gezielt bestimmt werden:

```
$auswahl += 2; // 0010 Rechner erhält eine Netzwerkkarte
$auswahl += 1; // 0011 Rechner erhält eine zusätzliche HD
$auswahl += 4; // 0111 Rechner erhält DVD-Brenner
$auswahl += 8; // 1111 Rechner erhält TV-Karte
```

Ergebnis:

Der Rechner ist voll ausgerüstet.

Der Vorteil besteht nun darin, dass man natürlich auch Komponenten entfernen kann.

```
$auswahl -= 4;  // 1011 DVD-Brenner wird entfernt
$auswahl -= 2;  // 1001 Netzwerkkarte wird entfernt
```

Ergebnis:

Der Rechner enthält lediglich eine »extra HD« und eine TV-Karte.

Nun sollten Sie sich die praktische Umsetzung betrachten. Um in PHP bitweise zu programmieren, kommen Sie um die Bitwise-Operatoren nicht herum.

Beispiel:

```
<?
// Alle Komponenten ausgewählt (kompletter Rechner)
$extraHD =     (1<<0); // 1 Bit: 0 (false), 1 (true)
$netzkarte =   (1<<1);         // 2 Bit: 0 (false), 2 (true)
$brenner =     (1<<2);         // 3 Bit: 0 (false), 4 (true)
$tvkarte =     (1<<3);         // 4 Bit: 0 (false), 8 (true)

// Die Komponenten in die Auswahl ablegen (Ergebnis 15)
$auswahl = $extraHD | $netzkarte | $brenner | $tvkarte;

// Hier nun eine Funktion, die den Preis berechnet
function berechne($auswahl) {
       $preis = 0;
       // Wenn das erste Bit gesetzt wurde, 200 Euro
       if ($auswahl & 1) {
              echo "+ Extra HD";
              $preis += 200;
       }
       // Wenn das zweite Bit gesetzt wurde, 150 Euro
       if ($auswahl & 2) {
              echo "+ Netzwerkkarte";
              $preis += 150;
       }
       // Wenn das dritte Bit gesetzt wurde, 450 Euro
       if ($auswahl & 4) {
              echo "+ DVD Brenner";
              $preis += 450;
```

```
        }
        // Wenn das vierte Bit gesetzt wurde, 100 Euro
        if ($auswahl & 8) {
                echo "+ TV-Karte";
                $preis += 100;
        }
        return $preis;
}

// Nun testen Sie die Umsetzung
echo berechne($auswahl);

/*
 Ausgabe:
 + Extra HD
 + Netzwerkkarte
 + DVD Brenner
 + TV-Karte
 900
*/

// Lediglich extraHD und tvkarte ausgewählt
$extraHD =     (1<<0); // 1 Bit: 0 (false), 1 (true)
$netzkarte = (0<<1);  // 2 Bit: 0 (false), 2 (true)
$brenner =     (0<<2);        // 3 Bit: 0 (false), 4 (true)
$tvkarte =     (1<<3);        // 4 Bit: 0 (false), 8 (true)

// Die Komponenten in die Auswahl ablegen (Ergebnis 9)
$auswahl = $extraHD | $netzkarte | $brenner | $tvkarte;

// Nun testen Sie die Umsetzung
echo berechne($auswahl);

/*
 Ausgabe:
 + Extra HD
 + TV-Karte
 300
*/

?>
```

Wir hoffen, Ihnen mit diesem Fallbeispiel einen Einblick in die Arbeit der Bitwise-Operatoren verschafft zu haben. Sie müssen natürlich selbst entscheiden, wie weit Sie sie in Ihre Anwendungen einbinden wollen.

Auflistung der Bitwise-Operatoren

Operator	*Bezeichnung*	*Bedeutung*
&	And	$a & $b – Bits, die in $a und $b gesetzt sind, werden gesetzt.
|	Or	$a | $b – Bits, die in $a oder $b gesetzt sind, werden gesetzt.
^	Xor	$a ^ $b – Bits, die entweder in $a oder $b gesetzt sind, werden gesetzt, aber nicht in beiden.

Operator	*Bezeichnung*	*Bedeutung*
~	Not	~ $a – Bits, die in $a nicht gesetzt sind, werden gesetzt, und umgekehrt.
<<	Shift left	$a << $b – Verschiebung der Bits von $a um $b Stellen nach links (jede Stelle entspricht einer Multiplikation mit zwei).
>>	Shift right	$a >> $b – Verschiebt die Bits von $a um $b Stellen nach rechts (jede Stelle entspricht einer Division durch zwei).

1.6.10 String-Operator

Der String-Operator ('.'), auch Zeichenkettenoperator genannt, nimmt eine Sonderstellung unter den Operatoren ein. Mit ihm lassen sich Strings zusammenfügen bzw. verbinden.

```
$a = "Hallo ";
$b = $a . "Welt !"; // Ergebnis $b = "Hallo Welt !"
```

Strings lassen sich mit Hilfe des String-Operators auch über mehrere Zeilen hinweg beschreiben.

```
$a = "Hallo ";
$a .= "Welt !"; // Ergebnis: $a = "Hallo Welt !"
```

1.6.11 Konditionaloperator

Der Konditionaloperator (? :) ist der einzige Operator, welcher drei Operanden benötigt. Der erste steht vor dem ?, der zweite zwischen ? und : und der dritte hinter dem :. Er wird übrigens auch als Bedingungsoperator bezeichnet, da man ihn für die Erzeugung einer einfachen If-Else-Anweisung nutzen kann.

(BEDINGUNG) ? AUSDRUCK1 : AUSDRUCK2;

Beispiel:

```
<?
$kAlter = 10;

// if-else Umsetzung
if ($kAlter >= 18) {
       echo "Erwachsener";
} else {
       echo "Zu jung!";
}

// Konditionaloperator (?:) Umsetzung
($kAlter >= 18) ? print "Erwachsener" : print "Zu jung!";
?>
```

Da der Operator die Ausdrücke nicht nur ausführt, sondern auch die Ergebnisse der Ausdrücke zurückliefert, können Sie ihn auch dazu verwenden, einer Variablen, in Abhängigkeit von einer Bedingung, unterschiedliche Werte zuzuweisen.

Beispiel:

```
// Initialisierung
$besuchName = "Matze";

$besuch = ($besuchName == "") ? "Hallo !" : "Hallo $besuchName";

echo $besuch; // Ausgabe: Hallo Matze
```

Hinweis: Die ?:-Syntax ist zwar sehr kurz und prägnant, sie ist aber auch relativ schwer zu lesen. Setzen Sie diese daher sparsam und nur in Fällen ein, wo sie leicht zu verstehen ist. Die Möglichkeit, komplexe Bedingungen aus mehreren Teilausdrücken zu verwenden, ist zwar gegeben, aber nicht ratsam.

1.6.12 Gruppierungsoperator

Der Gruppierungsoperator ( ) wird verwendet, um bevorzugte Operationen zusammenzufassen, oder dient bei Funktionen zum Übergeben von Parametern (Argumenten). Zusätzlich hat er die Aufgabe, Ausdrücke zu gruppieren, dabei kommt er vor allem bei `If`-Anweisungen zum Einsatz. Eine Besonderheit ist übrigens, dass er keine feste Anzahl von Operanden hat.

Beispiel:

```
// Berechnung (Addtion/Multiplikation)
$resultat = (1 + 2) * (3 + 4);              // Ergebnis: 21
$resultat = (1 + 2) * 3 + 4;                // Ergebnis: 13
$resultat = 1 + (2 * 3) + 4;                // Ergebnis: 11
$resultat = 1 + (2 * (3 + 4));              // Ergebnis: 15
```

Beispiel:

```
// Funktion (Parameter)
echo $resultat = sin(90);                   // Ergebnis: 0.89
```

Achtung: Er bestimmt die Reihenfolge, in der die Operatoren im Ausdruck ausgeführt werden. Runde Klammern setzen die automatische Vorrangreihenfolge außer Kraft und bewirken, dass die Ausdrücke in Klammern zuerst ausgewertet werden. Bei verschachtelten Klammern wird der Inhalt der innersten Klammern vor dem Inhalt der äußeren Klammern ausgewertet.

1.6.13 Inkrement- bzw. Dekrementoperatoren

PHP unterstützt Prä- und Post-Inkrement- und Dekrementoperatoren im Stil der Programmiersprache C.

Zwei in der Programmierung häufig benötigte Operationen sind die Erhöhung bzw. Verminderung eines Zahlenwerts um 1.

- Die Erhöhung um 1 bezeichnet man als Inkrement.
- Die Verminderung um 1 als Dekrement.

Für Inkrement und Dekrement gibt es in PHP zwei spezielle Operatoren:

- ++ (Inkrement)
- -- (Dekrement)

Beide Operatoren weisen gegenüber den anderen arithmetischen Operatoren einige Besonderheiten auf:

- Sie haben nur einen Operanden,
- können ihrem Operanden vor- oder nachgestellt werden (Präfix/Postfix),
- verändern den Wert ihres Operanden.

Folgendes Beispiel: Sie wollen den Wert einer Variablen i um 1 vermindern. Ohne Dekrementoperator würden Sie dafür schreiben:

```
$i= $i - 1;
```

Mit dem Dekrementoperator geht es schneller:

```
$i--;
```

Statt der Postfixnotation, dabei wird der Operator seinem Operanden nachgestellt, können Sie auch die Präfixnotation, hier ist der Operator seinem Operanden vorangestellt, verwenden.

```
--$i;
```

Sofern Sie den Dekrement- oder Inkrementoperator allein verwenden, ist es gleich, ob Sie die Postfix- oder Präfixnotation verwenden. Wenn Sie den Dekrement- oder Inkrementoperator in einem Ausdruck verwenden, müssen Sie jedoch klar zwischen Postfix- und Präfixnotation unterscheiden, denn beide führen zu unterschiedlichen Ergebnissen.

Beispiel:

```
$summe = 0;
$zahl = 20;
$summe = ++$zahl;      // Ergebnis: $summe  und $zahl  gleich 21
```

Hier wird der Wert der Variablen `$zahl` um 1 hochgesetzt und der neue Wert wird der Variablen `$summe` zugewiesen. Nach Ausführung der Anweisung sind `$summe` und `$zahl` gleich. Anders sieht es aus, wenn Sie den Operator nachstellen.

Beispiel:

```
$summe = 0;
$zahl = 20;
$summe = $zahl++;      // Ergebnis: $summe  20 und $zahl  21
```

Hier wird ebenfalls der Wert der Variablen `$zahl` um 1 hoch gesetzt, doch der Variablen `$summe` wird noch der alte Wert zugewiesen. Nach Ausführung der Anweisung hat `$summe` den Wert 20, während `$zahl` den Wert 21 hat. Ein weiteres Beispiel soll dies mit Hilfe einer `If`-Anweisung veranschaulichen.

Beispiel:

```
if (++$gehalt >= 2000) {
        ...
}
```

Im Beispiel mit der Präfixnotation wird der Wert der Variablen `$gehalt` zuerst um 1 erhöht und anschließend mit der Zahl 2000 verglichen.

Beispiel:

```
if ($gehalt++ >= 2000) {
        ...
}
```

Im Beispiel mit der Postfixnotation wird der Wert der Variablen `$gehalt` zuerst mit der Zahl 2000 verglichen und anschließend um 1 erhöht. Auch hier gibt es einen Unterschied, den es zu beachten gilt.

Operator	*Bezeichnung*	*Bedeutung*
++	Präinkrement	++$a ergibt $a+1 und erhöht $a um 1
++	Postinkrement	$a++ ergibt a und erhöht $a um 1
--	Prädekrement	--$a ergibt $a–1 und verringert $a um 1
--	Postdekrement	$a-- ergibt $a und verringert $a um 1

1.6.14 Objekterzeugungs-Operator

Der Operator `new` wird durch ein Schlüsselwort dargestellt und nicht durch Sonderzeichen. Es handelt sich hier um einen Operator, der vor seinem Operanden steht.

```
new Konstruktor
```

Konstruktur muss ein Funktionsaufruf-Ausdruck sein, d.h., es muss ein Ausdruck darin vorkommen, der sich auf eine Funktion bezieht, sogar auf eine ganze spezielle Funktion.

Beispiel:

```
<?php
// Klasse
class Haus
{
        var $zimmer;
        function Haus($zimmer)
        {
                $this->zimmer = $zimmer;
```

```
        }
}

// Objekt
$meinHaus = new Haus(8);

// Ausgabe - Object id #1
echo $meinHaus;
// Ausgabe (8)
echo $meinHaus->zimmer;
?>
```

Der Operator `new` funktioniert wie folgt: Zuerst wird ein neues Objekt ohne jegliche Eigenschaften angelegt und anschließend wird die angegebene Konstruktorfunktion mit den angegebenen Parametern aufgerufen.

1.6.15 Array-Operatoren

Der einzige Array-Operator in PHP ist der +-Operator. Das rechtsstehende Array wird an das linksstehende Array angehängt, wobei doppelte Schlüssel NICHT überschrieben werden.

Beispiel:

```
<pre>
<?php
$personen = array("a" => "Matthias", "b" => "Carolline");
$fruechte = array("a" =>"Kirsche", "b" => "Erdbeere", "c" => "Birne");
$gesamt = $personen + $fruechte;
var_dump($gesamt);
?>
</pre>
```

Ausgabe:

```
array(3) {
  ["a"]=>
  string(8) "Matthias"
  ["b"]=>
  string(9) "Carolline"
  ["c"]=>
  string(5) "Birne"
}
```

1.6.16 Operatoren zur Programmausführung

PHP unterstützt einen Operator zur Ausführung externer Programme: den Backtick (``). Achtung: Die Backticks sind keine einfachen Anführungszeichen! PHP versucht, den Text zwischen den Backticks als Kommandozeilenbefehl auszuführen. Die Ausgabe des aufgerufenen Programms wird zurückgegeben und kann somit einer Variablen zugewiesen werden.

```
<pre>
<?php
$ausgabe = `ls -al`;
echo $ausgabe;
?>
</pre>
```

Achtung: Der Backtick-Operator steht nicht zur Verfügung, wenn der Safe Mode aktiviert ist oder die Funktion `shell_exec()` deaktiviert wurde.

1.6.17 Fehler-Kontroll-Operatoren

PHP unterstützt einen Operator zur Fehlerkontrolle. Es handelt sich dabei um das @-Symbol. Stellt man @ in PHP vor einen Ausdruck, werden alle Fehlermeldungen, die von diesem Ausdruck erzeugt werden könnten, ignoriert.

Hinweis: Ist das track_errors-Feature aktiviert, werden alle Fehlermeldungen, die von diesem Ausdruck erzeugt werden, in der Variablen `$php_errormsg` gespeichert. Da diese Variable mit jedem neuen Auftreten eines Fehlers überschrieben wird, sollte man sie möglichst bald nach Verwendung des Ausdrucks überprüfen, wenn man mit ihr arbeiten will.

Beispiel:

```
<?php
// Beabsichtigter Dateifehler
// Beabsichtigter Mailfehler
$email = @mail ('nicht_vorhandene_mail') or
    die ("Mail konnte nicht versandt werden:'$php_errormsg'");

// Das funktioniert bei jedem Ausdruck
// erzeugt keine Notice, falls der Index
// $key nicht vorhanden ist.
$value = @$cache[$key];
?>
```

Der Fehler-Kontroll-Operator @ verhindert jedoch keine Meldungen, welche aus Fehlern beim Parsen resultieren.

Achtung: Der @-Operator funktioniert nur bei Ausdrücken. Eine einfache Daumenregel: Wenn Sie den Wert von etwas bestimmen können, dann können Sie den @-Operator davor schreiben. Zum Beispiel können Sie ihn vor Variablen, Funktionsaufrufe und vor `include()` setzen, vor Konstanten und so weiter. Nicht verwenden können Sie diesen Operator vor Funktions- oder Klassendefinitionen oder vor Kontrollstrukturen wie zum Beispiel `if` und `foreach` und so weiter.

Achtung: Zum gegenwärtigen Zeitpunkt deaktiviert der Fehler-Kontroll-Operator @ die Fehlermeldungen selbst bei kritischen Fehlern, die die Ausführung eines Skripts beenden. Unter anderem bedeutet das, wenn Sie @ einer bestimmten Funktion voranstellen, diese aber nicht zur Verfügung steht oder falsch geschrieben wurde, Ihr PHP-Skript einfach beendet wird, ohne Hinweis auf die Ursache.

1.7 Kontrollstrukturen

Die Kontrollstrukturen haben die Aufgabe, den Ablauf eines Programms zu beeinflussen. Sie sind in der Lage, die Programmausführung in Abhängigkeit von einer Bedingung zu steuern oder einzelne Anweisungen oder Anweisungsblöcke wiederholt auszuführen. In diesem Abschnitt soll Ihnen die Arbeitsweise der Kontrollstrukturen näher gebracht werden. Sie werden dabei feststellen, dass PHP eine Vielzahl von Steuerungsmöglichkeiten besitzt.

1.7.1 if-Anweisung

Die `if`-Anweisung dient dazu, anhand einer Bedingung, z.B. des Werts einer Variablen, des Rückgabewerts einer Funktion oder des Wahrheitswerts eines booleschen Werts, zu entscheiden, ob die nachfolgende Anweisung ausgeführt werden soll oder nicht.

Definition:

IF (BEDINGUNG/AUSDRUCK)
    ANWEISUNG;

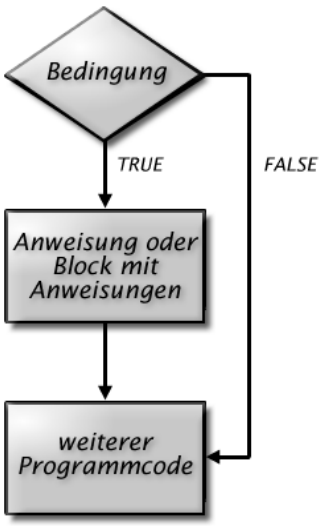

Bild 1.3: Ablaufschema einer `if`-Anweisung

Beispiel:

```
$punkteStand = 100;

// Schreibweise
if ($punkteStand == 100)
        echo "Vorhanden!";

// Schreibweise (Ebenfalls möglich)
if ($punkteStand == 100) echo "Vorhanden!";
```

Sie können die `if`-Anweisung wie folgt übersetzen:

»Wenn (if) die Bedingung erfüllt (true liefert, also die Variable `$punkteStand` existiert), dann führe die Anweisung aus (Variable kommt zustande mit dem Wert »Vorhanden!«). Ist die Bedingung nicht erfüllt, führe die Anweisung nicht aus.«

Hinweis: Die `if`-Anweisung entscheidet über die Ausführung der direkt nachfolgenden Anweisung bzw. des nachfolgenden Anweisungsblocks. Üblicherweise rückt man die betreffenden Anweisungen im Quelltext um zwei bis drei Leerzeichen ein. Dies dient jedoch ausschließlich der besseren Lesbarkeit des Codes, auf die Ausführung der `if`-Anweisung hat die Einrückung keinen Einfluss.

Wenn Sie mit einer `if`-Anweisung nicht nur die Ausführung einer einzelnen, sondern mehrerer aufeinander folgender Anweisungen steuern wollen, müssen Sie die betreffenden Anweisungen in geschweifte Klammern setzen.

Definition:

IF (BEDINGUNG/AUSDRUCK) {
        ANWEISUNG/EN;
}

```
$punkteStand = 100;

// Anweisungsblock
if ($punkteStand == 100) {
        echo "Vorhanden!";
        $bonus = 50;
}
```

Hinweis: Eine Folge von Anweisungen, die in geschweifte Klammern eingefasst sind, bezeichnet man als Anweisungsblock. Einige Programmierer würden dies auch als zusammengesetzte Anweisungen bezeichnen. Es bleibt Ihnen überlassen, wie Sie das Gebilde nennen.

Alternative Syntax

In PHP steht Ihnen für die `if`-Anweisung eine alternative Schreibweise zur Verfügung.

Definition:

IF (BEDINGUNG/AUSDRUCK): ANWEISUNG; ENDIF;

Beispiel – alternative Schreibweise:

```
$vorname = "Caroline";
if ($vorname == "Caroline"):
        echo "Name: $vorname";
        echo "Mehrzeilig!";
endif;
```

Verwechselung von == und = vermeiden

Um Verwechselungen beim Durchführen von Vergleichen innerhalb von `if`-Anweisungen zu vermeiden, sollten Sie sich folgenden Rat zu Herzen nehmen:

Schreiben Sie anstatt

```
<?
$mitarbeiter = 10;
if ($mitarbeiter == 10) { echo "10 Mitarbeiter"; }
?>
```

besser:

```
<?
$mitarbeiter = 10;
if (10 == $mitarbeiter) { echo "10 Mitarbeiter"; }
?>
```

Denn sollten Sie die Konstante auf der linken Seite mit einem Zuweisungsoperator versehen, wird der PHP-Interpreter eine Fehlermeldung ausgeben.

```
<?
$mitarbeiter = 10;
// Führt zu:
// parse error, unexpected '='
if (10 = $mitarbeiter) { echo "10 Mitarbeiter"; }
?>
```

Die Gefahr, durch einen Eingabefehler falsche Werte zu erhalten, verringert sich dadurch und Sie können relativ sicher sein, innerhalb von `if`-Anweisungen keine falsch platzierten Zuweisungen vorliegen zu haben.

Hinweis: Diese Schreibweise hat sich in vielen Fällen jedoch nicht durchgesetzt. Es scheint eine Frage des Geschmacks zu sein, da eine Vielzahl von PHP-Entwickler doch lieber ihrem eigenem Können vertrauen.

1.7.2 if-else-Anweisung

Die einfache `if`-Anweisung entspricht der Anweisung »Wenn die Bedingung erfüllt ist, dann führe die Anweisung aus. Danach fahre mit der Ausführung des Programms fort.«

Die zweite Form der `if`-Anweisung führt den `else`-Teil ein, der ausgeführt wird, wenn die Bedingung nicht erfüllt, also der Ausdruck `false` ist. Die Aussage der Anweisung lautet dann: »Wenn die Bedingung erfüllt ist, dann führe die Anweisung aus, sonst (`else`) führe die Anweisung im erweiterten Teil aus. Danach fahre normal mit der Ausführung des Programms fort.«.

Definition:

IF (BEDINGUNG/AUSDRUCK)
    ANWEISUNG;
ELSE
    ANWEISUNG;

Beispiel:

```
$punkteStand = 50;

// Schreibweise
if ($punkteStand >= 100)
    echo "Wahr";
else
    echo "Unwahr!";
```

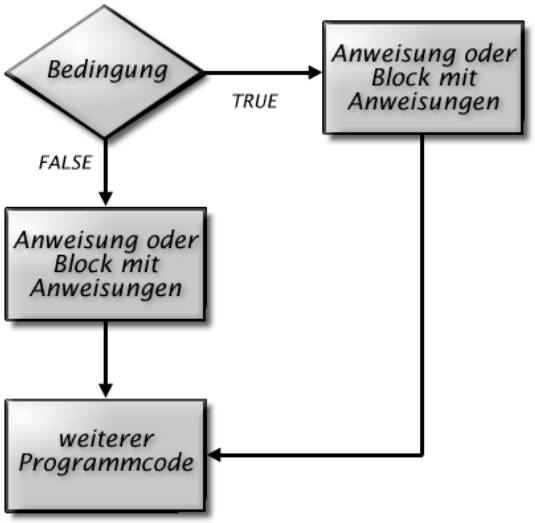

Bild 1.4: Ablaufschema einer if-else-Anweisung

Wie sieht es nun mit einem Anweisungsblock in einer `if-else`-Anweisung aus? Auch das stellt kein Problem dar:

Definition:

IF (BEDINGUNG/AUSDRUCK) {
    ANWEISUNG/EN;
} ELSE {
    ANWEISUNG/EN;
}

```
$punkteStand = 50;

// Anweisungsblock
if ($punkteStand >= 100) {
        echo "Wahr";
} else {
        echo "Unwahr!";
}
```

Alternative Syntax

In PHP steht Ihnen für die `if-else`-Anweisung eine alternative Schreibweise zur Verfügung.

Definition:

IF (BEDINGUNG/AUSDRUCK): ANWEISUNG/EN; ELSE: ANWEISUNG/EN; ENDIF;

Beispiel – bekannte Schreibweise:

```
<?php
$chef = "Schmidt";
if ($chef == "Schmidt") {
        echo "Chef ist Schmidt";
} else {
        echo "Unbekannter Chef";
}
?>
```

Beispiel – alternative Schreibweise:

```
<?
$chef = "Schmidt";
if ($chef == "Schmidt"):
?>
Chef ist Schmidt
<?
else:
?>
Unbekannter Chef
<?
endif
?>
```

Eine weitere Möglichkeit wäre, die alternative Schreibweise wie folgt zu verwenden.

```
$signal = TRUE;
if ($signal):
        echo "Aktiv: $signal";
        echo "Mehrzeilig!";
else:
        echo "Inaktiv";
        echo "Mehrzeilig";
endif;
```

Mit der Einführung des Schlüsselworts `endif` kann das Ende des Blocks explizit angegeben werden. Der `else`-Befehl ist mit einem Doppelpunkt zu versehen, um die Zugehörigkeit zum Block zu kennzeichnen. Die geschweiften Klammern entfallen.

1.7.3 if-elseif-Anweisung

Die `if-elseif`-Anweisung wird vor allem für Mehrfachverzweigungen eingesetzt. Damit lassen sich in Abhängigkeit vom Wert einer Variablen verschiedene Anweisungen ausführen.

Definition:

IF (BEDINGUNG/AUSDRUCK) {
        ANWEISUNG/EN;
} ELSEIF (BEDINGUNG/AUSDRUCK) {
        ANWEISUNG/EN;
} ELSEIF (BEDINGUNG/AUSDRUCK) {
        ANWEISUNG/EN;
} ELSE {
        ANWEISUNG/EN;
}

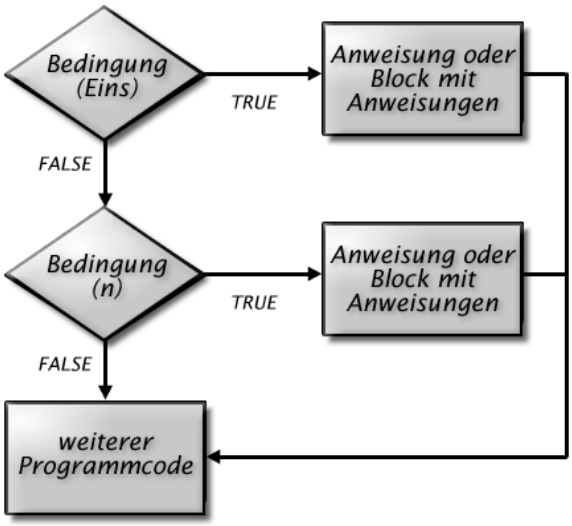

Bild 1.5: Ablaufschema einer if-elseif-Anweisung

Fallbeispiel:

WENN MEINLOHN DEN WERT
1000 HAT: FÜHRE ANWEISUNG A AUS,
2000 HAT: FÜHRE ANWEISUNG B AUS,
3000 HAT: FÜHRE ANWEISUNG C AUS,
4000 HAT: FÜHRE ANWEISUNG D AUS,
IN ALLEN ANDERN FÄLLEN: FÜHRE ANWEISUNG E AUS.

Beachten Sie den grundsätzlichen Unterschied zur `if-else`-Anweisung. Die Bedingung einer `if-else`-Anweisung ist immer ein boolescher Wert, der nur einen der beiden Werte `true` oder `false` annehmen kann. Folglich verzweigt die `if-else`-Anweisung auch nur in zwei alternative Anweisungsblöcke. Bei der Mehrfachverzweigung wird dagegen der Wert einer Variablen abgefragt: Der Programmierer kann grundsätzlich ebenso viele alternative Verzweigungen formulieren, wie es Werte für die Variable gibt.

```
$meinLohn = 3000;

// Mehrfachverzweigung (if-elseif-Anweisung)
// Ergebnis: "C"
if ($meinLohn == 1000) {
       // Anweisung A
       $ausgabe = "A";
} elseif ($meinLohn == 2000) {
       // Anweisung B
       $ausgabe = "B";
} elseif ($meinLohn == 3000) {
       // Anweisung C
       $ausgabe = "C";
} elseif ($meinLohn == 4000) {
       // Anweisung D
       $ausgabe = "D";
} else {
       // Anweisung E
       $ausgabe = "E";
}

echo $ausgabe;
```

Lassen Sie sich aber nicht von der Einrückung täuschen. Es handelt sich hier immer noch um vier, immer tiefer verschachtelte `if-else`-Anweisungen. Der Interpreter prüft zuerst, ob der Wert von `meinLohn` gleich 1.000 ist. Ist dies nicht der Fall, prüft er im `else`-Teil, ob `meinLohn` gleich 2.000 ist. Stimmt auch dies nicht, verzweigt er zum `else`-Teil mit dem Vergleich `meinLohn` gleich 3.000. Stimmt auch dies nicht, verzweigt er zum `else`-Teil mit dem Vergleich `meinLohn` gleich 4.000. Liefert auch dieser Vergleich `false`, landet der Interpreter in dem letzten `else`-Teil, der alle anderen nicht überprüften Fälle abfängt.

Alternative Syntax

In PHP steht auch für die `if-elseif`-Anweisung eine alternative Schreibweise zur Verfügung.

Definition:

IF (BEDINGUNG/AUSDRUCK):
    ANWEISUNG/EN;
ELSEIF:
    ANWEISUNG/EN;
ELSE:
    ANWEISUNG/EN;
ENDIF;

Beispiel – bekannte Schreibweise:

```
<?php
$chef = "Schmidt";
if ($chef == "Müller") {
        echo "Müller ist der Chef";
} elseif ($chef == "Schmidt") {
        echo "Scmidt ist der Chef";
} else {
        echo "Unbekannter Chef";
}
?>
```

Beispiel – alternative Schreibweise:

```
<?
$chef = "Schmidt";
if ($chef == "Müller"):
?>
Müller ist der Chef
<?
elseif ($chef == "Schmidt"):
?>
Schmidt ist der Chef
<?
else:
?>
Unbekannter Chef
<?
endif
?>
```

1.7.4 switch-case-Anweisung

Eine Alternative zur `if-elseif`-Anweisung stellt die `switch`-Anweisung dar. Sie ermöglicht ebenfalls eine Mehrfachverzweigung, jedoch wirkt der Code wesentlich übersichtlicher und trägt zum besseren Verständnis bei.

Definition:

```
SWITCH (BEDINGUNG/AUSDRUCK){
        CASE WERT:
                ANWEISUNG/EN;
        CASE WERT:
                ANWEISUNG/EN;
        [DEFAULT WERT:]
                [ANWEISUNG/EN;]
}
```

```
$meinLohn = 3000;

// Mehrfachverzweigung (switch-Anweisung)
// Ergebnis: "C"
switch ($meinLohn) {
        case 1000:
                // Anweisung A
                $ausgabe = "A";
                break;
        case 2000:
                // Anweisung B
                $ausgabe = "B";
                break;
        case 3000:
                // Anweisung C
                $ausgabe = "C";
                break;
        case 4000:
                // Anweisung D
                $ausgabe = "D";
                break;
        default:
                // Anweisung E
                $ausgabe = "E";
}

echo $ausgabe;
```

Es dreht sich auch hier wieder alles um den Ausdruck, der in der Regel eine Variable ist. Der `default`-Teil wird ausgeführt, wenn keiner der vorherigen Werte zutrifft. Dieser Teil der `switch`-Anweisung ist optional. Jeder Programmblock muss mit dem Kommando `break` abgeschlossen werden, damit die `switch`-Anweisung nicht unnötig weiter durchlaufen wird, wenn es zu einer Übereinstimmung mit einer der `case`-Werte kommt.

Hinweis: `break` bricht die Ausführung der aktuellen Anweisungssequenz `for`, `foreach` `while`, `do..while` oder `switch` ab.

Erweiterte Syntax

Die Beschränkung des Bedingungstests auf Gleichheit mag als ernsthafte Behinderung erscheinen. Glücklicherweise kennt PHP eine erweiterte Notation der Syntax, die das Problem teilweise behebt. Dabei können mehrere Vergleiche hintereinander ausgeführt werden, die Operanden sind quasi Oder-verknüpft:

```
<?php
$meinLohn = 3000;

// Mehrfachverzweigung (switch-Anweisung)
switch ($meinLohn) {
        case 1000: case 2000: case 3000:
                // Anweisung A
                $ausgabe = "Zwischen 1000 u. 3000";
                break;
        case 4000: case 5000:
                // Anweisung D
                $ausgabe = "Zwischen 4000 u. 5000";
                break;
        default:
                // Anweisung E
                $ausgabe = "Kein Angabe";
}
// Ausgabe - Zwischen 1000 u. 3000
echo $ausgabe;
?>
```

Alternative Syntax

Definition:

SWITCH (BEDINGUNG/AUSDRUCK):
        CASE WERT:
                ANWEISUNG/EN;
        DEFAULT:
                ANWEISUNG/EN;
ENDSWITCH;

Hier ein Beispiel für die alternative Schreibweise einer `switch`-Anweisung:

```
<?
$meinLohn = 3000;

// Mehrfachverzweigung (switch-Anweisung)
// Ergebnis: "C"
switch ($meinLohn):
        case 1000:
                // Anweisung A
                $ausgabe = "A";
                break;
        case 2000:
                // Anweisung B
```

```
                $ausgabe = "B";
                break;
        case 3000:
                // Anweisung C
                $ausgabe = "C";
                break;
        case 4000:
                // Anweisung D
                $ausgabe = "D";
                break;
        default:
                // Anweisung E
                $ausgabe = "E";
endswitch;

echo $ausgabe;
?>
```

1.7.5 while-Schleife

Während die `if`-Anweisung die grundlegende Steueranweisung ist, mit deren Hilfe in PHP Entscheidungen gefällt werden, können mit dem Sprachelement `while` Anweisungen wiederholt ausgeführt werden.

Definition:

WHILE (BEDINGUNG/AUSDRUCK) {
    ANWEISUNG/EN;
}

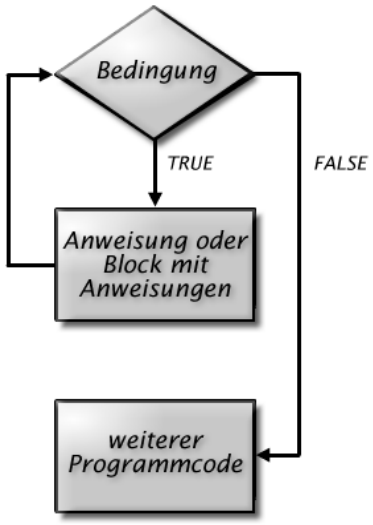

Bild 1.6: Ablaufschema einer while-Schleife

Beispiel:

```
// Laufvariable (Zählvariable)
$i = 0;

// while-Anweisung
// Ausgabe: 0 1 2 3 4 5 6 7 8 9
while ($i < 10) {              // Bedingung
       echo "$i<br>";          // Anweisungsblock
       $i++;                   // Inkrementierung der Laufvariablen
}
```

Und so funktioniert die `while`-Anweisung: Zunächst wird der angegebene Ausdruck berechnet. Ist er `false`, geht der Interpreter zur nächsten Anweisung im Programm über. Sollte diese jedoch `true` sein, wird die angegebene Anweisung bzw. der entsprechende Anweisungsblock ausgeführt. Anschließend wird der Ausdruck neu berechnet. Wiederum geht der Interpreter zur im Programm folgenden Anweisung über, falls der Wert des Ausdrucks `false` sein sollte. Ansonsten wird erneut die Anweisung ausgeführt, die den Hauptteil der Schleife ausmacht. Dieser Kreislauf wird so lange durchlaufen, bis der Ausdruck schließlich `false` ergibt. Dann wird die Ausführung der `while`-Anweisung beendet, und der PHP-Interpreter fährt mit der sequentiellen Abarbeitung des Programms fort.

Hinweis: Schleifendurchläufe bezeichnet man auch als Iterationen.

Hier noch ein Beispiel, mit dessen Hilfe Sie die Zeichen der ASCII-Tabelle im Handumdrehen erzeugt haben.

```
$zaehler = 32;
while ($zaehler < 127) {
       $zeichen = chr($zaehler);
       echo $zeichen . " |";
       $zaehler++;;
}
```

Ausgabe:

```
|! |" |# |$ |% |& |' |( |) |* |+ |, |- |. |/ |0 |1 |2 |3 |4 |5 |6 |7 |8 |9
|: |; |< |= |> |? |@ |A |B |C |D |E |F |G |H |I |J |K |L |M |N |O |P |Q |R
|S |T |U |V |W |X |Y |Z |[ |\ |] |^ |_ |` |a |b |c |d |e |f |g |h |i |j |k
|l |m |n |o |p |q |r |s |t |u |v |w |x |y |z |{ || |} |~ |
```

Alternative Syntax

Auch für die `while`-Schleife existiert in PHP eine alternative Schreibweise.

Definition:

```
WHILE (BEDINGUNG/AUSDRUCK):
       ANWEISUNG/EN;
ENDWHILE;
```

Beispiel – bekannte Schreibweise:

```
<?php
$zaehler = 0;
$max = 10;
while ($zaehler < $max) {
       if ($zaehler % 2) {
              $zaehler++;
              continue;
       }
       echo "Zaehler: $zaehler <br>";
       $zaehler++;
}
?>
```

Beispiel – alternative Schreibweise:

```
<?
$zaehler = 0;
$max = 10;
while ($zaehler < $max):
if ($zaehler % 2) {
 $zaehler++;
 continue;
}
?>
Zaehler: <? echo $zaehler;?><br>
<? $zaehler++; ?>
<? endwhile ?>
```

1.7.6 do-while-Schleife

Die `do-while`-Anweisung (Schleife) ist der `while`-Anweisung sehr ähnlich. Der einzige Unterschied besteht darin, dass die Schleifenbedingung nicht am Anfang, sondern am Ende der Schleife überprüft wird. Deshalb führt die `do-while`-Anweisung mindestens einen Durchlauf aus.

Definition:

DO {
    ANWEISUNG/EN;
} WHILE (BEDINGUNG/AUSDRUCK);

Beispiel:

```
// Laufvariable (Zählvariable)
$i = 0;

// do-while-Anweisung
// Ausgabe: 0 1 2 3 4 5 6 7 8 9
do {
    echo "$i<br>";   // Anweisungsblock
    $i++;            // Inkrementierung der Laufvariablen
} while ($i < 10);   // Bedingung
```

Hinweis: Für `do...while` können Sie keine alternative Schreibweise anwenden.

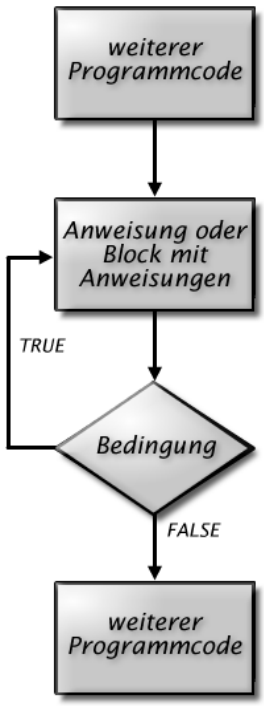

Bild 1.7: Ablaufschema einer do-while-Schleife

1.7.7 for-Schleife

Die `for`-Anweisung ist für Schleifen oft praktischer als die `while`-Anweisung. Hier wird ausgenutzt, dass die meisten Schleifen einem bestimmten Schema folgen. Normalerweise gibt es eine Schleifenvariable, die vor Beginn der Schleife initialisiert wird. Vor jedem Schleifendurchlauf wird der Wert dieser Variablen innerhalb des Ausdrucks überprüft. Schließlich wird sie am Ende der Schleife, unmittelbar vor der erneuten Auswertung des Ausdrucks, inkrementiert oder in anderer Weise geändert. Sie hat des Weiteren den Vorzug, dass die Initialisierung, der Ausdruck (Bedingung) und die Veränderung der Schleifenvariablen übersichtlich im Kopf der Schleife zusammengefasst sind.

Definition:

```
FOR (INIT SCHLEIFENVARIABLE; BEDINGUNG; VERÄNDERUNG) {
        ANWEISUNG/EN;
}
```

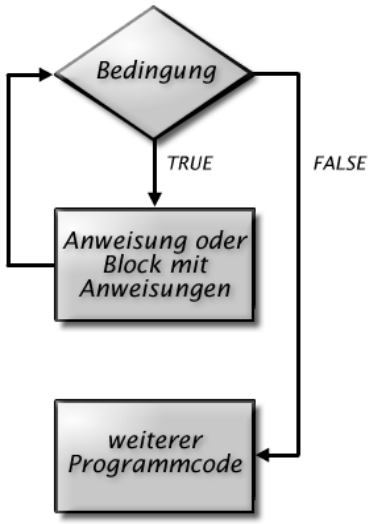

Bild 1.8: Ablaufschema einer for-Schleife

Am einfachsten lässt sich erklären, was die `for`-Anweisung bewirkt, wenn man die äquivalente `while`-Anweisung angibt:

```
// Laufvariable (Zählvariable)
$i = 0;

// while-Anweisung
// Ausgabe: 0 1 2 3 4 5 6 7 8 9
while ($i < 10) {              // Bedingung
       echo "$i<br>";          // Anweisungsblock
       $i++;                   // Inkrementierung der Laufvariablen
}
```

```
// for-Anweisung
// Im Ausgabefenster: 0 1 2 3 4 5 6 7 8 9
for ($i=0;$i<10;$i++) {
       echo "$i<br>";          // Anweisungsblock
}

// Schreibweise (Variante 2)
for ($i=0;$i<10;$i++)
       echo "$i<br>";          // Anweisung
```

Wie Sie hier lesen, stehen in dieser Schreibweise alle wichtigen Information über die Schleifenvariable in einer einzigen Codezeile. Dadurch ist die Arbeitsweise der Schleife klar ersichtlich. Außerdem wird dadurch, dass die Veränderung der Schleifenvariablen innerhalb der eigentlichen `for`-Anweisung steht, der Anweisungsblock vereinfacht.

Flexibilität der for-Schleife

Alle drei Parameter der `for`-Schleife sind äußerst flexibel einsetzbar. So sind die drei Parameter der `for`-Schleife optional. Ohne eine Iterationsvariable wird die Schleife endlos durchlaufen. Sie können in diesem Fall wieder auf `break` zurückgreifen, um die Schleife mit einer zusätzlichen Bedingung zu verlassen.

Beispiel:

```
<?php
$zaehler = 1;
for (;;) {
    if ($zaehler > 10) {
        break;
    }
    print $zaehler;
    $zaehler++;
}
?>
```

Hinweis: `continue` kann ebenfalls zum Einsatz kommen.

Der flexible Umgang mit den Schleifenparametern kennt praktisch keine Grenzen. Auch das folgende Beispiel ist syntaktisch korrekt:

```
<?php
for ($i = 0, $j = 10;$i<$j;$i++) {
    $j--;
    echo "i ist jetzt: $i<br>";
    echo "j ist jetzt: $j<br>";
}
?>
```

Ausgabe:

```
i ist jetzt: 0
j ist jetzt: 9
i ist jetzt: 1
j ist jetzt: 8
i ist jetzt: 2
j ist jetzt: 7
i ist jetzt: 3
j ist jetzt: 6
i ist jetzt: 4
j ist jetzt: 5
```

Es spielt offensichtlich keine Rolle, ob hier Variablen zum Einsatz kommen oder nicht. Wenn Sie beispielsweise lediglich eine Liste ausgeben wollen, können Sie den Befehl recht knapp halten:

```
<?php
for ($i = 1; $i <= 10; print $i, $i++) ;
?>
```

Innerhalb des `for`-Befehls können also weitere Befehle, durch Kommata getrennt, eingesetzt werden. Im Beispiel wird übrigens nicht zufällig der Befehl `print` anstatt des flexibleren `echo` verwendet. Wie bei der Befehlsbeschreibung bereits erläutert, gibt `echo` nichts zurück, während `print` die Ausführung im Erfolgsfall mit `TRUE` quittiert. Normalerweise wird der Rückgabewert nicht benötigt und gelöscht. Die `for`-Schleife erwartet jedoch von jedem direkt eingegebenen Wert, dass er sich als Ausdruck verhält – Ausdrücke geben immer etwas zurück. An dieser Stelle kann `echo` also nicht funktionieren. Versuchen Sie es dennoch, erhalten Sie einen Laufzeitfehler.

Alternative Syntax

Natürlich, wie kann es anders sein, gibt es auch für die `for`-Schleife eine alternative Schreibweise.

Definition:

```
FOR (INIT SCHLEIFENVARIABLE; BEDINGUNG; VERÄNDERUNG):
        ANWEISUNGEN/EN;
ENDFOR;
```

Beispiel – alternative Schreibweise:

```
<? for ($i = 0; $i <= 10; $i++): ?>
i ist jetzt: <? echo $i ?><br>
<? endfor; ?>
```

Ausgabe:

```
i ist jetzt: 0
i ist jetzt: 1
i ist jetzt: 2
i ist jetzt: 3
i ist jetzt: 4
i ist jetzt: 5
i ist jetzt: 6
i ist jetzt: 7
i ist jetzt: 8
i ist jetzt: 9
i ist jetzt: 10
```

1.7.8 foreach-Schleife

Diese Schleife ermöglicht es, auf einfache Weise ein Array zu durchlaufen. `foreach` funktioniert nur in Verbindung mit Arrays. Wenn Sie versuchen, `foreach` mit einer Variablen eines anderen Datentyps oder einer nicht initialisierten Variablen zu benutzen, gibt PHP einen Fehler aus.

Definition:

```
FOREACH (ARRAY_EXPRESSION AS $VALUE){
        ANWEISUNG/EN;
}
```

Mit Hilfe von `foreach` wird das Durchlaufen eines Arrays wesentlich vereinfacht. Gegenüber der `while`-Schleife mit `list` und `each` ist die `foreach`-Schleife syntaktisch deutlich im Vorteil.

Beispiel – while:

```
<?php
$zahlen = array (10, 20, 30, 40);
while (list(, $value) = each ($zahlen)) {
    echo "Wert: $value<br>\n";
}
?>
```

Beispiel – foreach:

```
<?php
foreach ($zahlen as $value) {
    echo "Wert: $value<br>\n";
}
?>
```

Das Array wird von Anfang bis Ende durchlaufen und bei jedem Schleifendurchlauf wird das aktuelle Element der Variablen `array_expression` zugewiesen. Jedes Arrayelement kann wiederum ein Array sein, dann könnte mit einer verschachtelten `foreach`-Schleife auch dieses Array ausgewertet werden.

Beispiel – mit assoziativem Array:

```
$personen = array("Matthias","Caroline","Gülten");

foreach ($personen as $person) {
    echo "Name: $person<br>\n";
}
// Ausgabe:
// Name: Matthias
// Name: Caroline
// Name: Gülten
```

Typischerweise ist es erlaubt, einzelne Elemente eines solchen Arrays mit unterschiedlichen Datentypen zu belegen. Das können auch weiere Arrays sein.

Erweiterte Syntax

Sollten Sie Arrays mit Schlüssel/Werte-Paaren bauen, kann `foreach` mit einer erweiterten Syntax diese Paare direkt auslesen. Die grundlegende Syntax lautet:

Definition:

```
FOREACH (ARRAY AS $KEY => $VALUE){
        ANWEISUNG/EN;
}
```

Hier wird der Operator => eingesetzt, der schon bei der Konstruktion des Arrays verwendet wurde.

Beispiel:

```
<?php
$person = array("Vorname" => "Caroline",
                            "Nachname" => "Kannengiesser",
                            "Alter" => 25,
                            "Ort" => Berlin);
foreach ($person as $key => $val) {
       echo "Feld $key hat den Wert: $val<br>";
}
?>
```

Ausgabe:

```
Feld Vorname hat den Wert: Caroline
Feld Nachname hat den Wert: Kannengiesser
Feld Alter hat den Wert: 25
Feld Ort hat den Wert: Berlin
```

Hinweis: Mehr zum Thema Arrays finden Sie im Abschnitt »Arrays«.

1.7.9 Verschachtelte Kontrollstrukturen

Sie haben nun alle Kontrollstrukturen aus PHP vorgestellt bekommen. Sie lassen sich nicht nur jeweils einzeln ausführen, sondern können auch miteinander kombiniert werden. Das Kombinieren von Kontrollstrukturen untereinander wird in der Programmierung als Verschachtelung bezeichnet.

Sie kennen sicher die aus Russland stammenden »Matuschkas«, das sind die ineinander geschachtelten Puppen. Jedesmal, wenn Sie eine Puppe öffnen, kommt in ihrem Inneren eine weitere, kleinere Puppe zum Vorschein. Dies wird fortgeführt, bis Sie zur letzten Puppe kommen, die sich nicht mehr öffnen lässt. Sie können sich verschachtelte Kontrollstrukturen so ähnlich aufgebaut vorstellen. Man schaut immer in die nächste hinein, dem Prinzip der Kiste in der Kiste entsprechend. Dabei kommt die Verschachtelung sowohl bei `if`-Anweisungen und `switch`-Anweisungen als auch bei Schleifen vor.

Beispiel:

```
<?php
/*
       Beispiel Haus (Verschachtelung: 3 Stufen)
       Aufbau der Stufen:
```

```
        // if-Stufe 1
                // if-Stufe 2
                        // if-Stufe 3
                        // else-Stufe 3
                // else-Stufe 2
        // else-Stufe 1
*/

$haus = true;
$hausTuer = true;
$hausSchluessel = true;

if ($haus == true) {
        echo "Haus in Sicht - Tür vorhanden?";
        if ($hausTuer == true) {
                echo "Tür vorhanden - passender Schlüssel?";
                if ($hausSchluessel == true) {
                        echo "Passender Schlüssel - Haus betreten!";
                } else {
                        echo "Leider keinen passenden Schlüssel!";
                }
        } else {
                echo "Leider hat das Haus keine Tür";
        }
} else {
        echo "Kein Haus in Sicht!";
}
?>
```

Ausgabe:

```
Haus in Sicht - Tür vorhanden?
Tür vorhanden - passender Schlüssel?
Passender Schlüssel - Haus betreten!
```

Tipp: Ein Limit für die Anzahl von Verschachtelung gibt es nicht. Aber je weniger Stufen die Verschachtelung verwendet, desto leichter ist der Code zu verstehen. Sie sollten bei der Verschachtelung die Faustregel beachten, nicht mehr als drei Stufen zu verwenden, da Sie sonst den Überblick verlieren könnten.

Bei der Verschachtelung von Kontrollstrukturen sollten Sie unbedingt darauf achten, dass Sie Ihre Codezeilen einrücken. Das wirkt sich vor allem besonders positiv auf die Lesbarkeit des Codes aus.

Beispiel:

```
<?php
// switch-Anweisung in der sich if-Anweisungen befinden.
$lohn = 1000;
$mitarbeiter = "Mike";

// Ausgabe - Mike du erhälst 1000 Euro pro Monat
switch ($lohn) {
```

```
        case 1000:
                if ($mitarbeiter == "Mike") {
                        echo "Mike du erhälst 1000 Euro pro Monat";
                }
                break;
        case 2000:
                if ($mitarbeiter == "Gülten") {
                        echo "Gülten du erhälst 2000 Euro pro Monat";
                }
                break;
        default:
                echo "Sie sind kein Mitarbeiter unserer Firma!";
}
?>
```

Sie sehen, eine Verschachtelung kann durch das Einrücken der Codezeilen um einiges besser überblickt werden.

Hinweis: Die Verschachtelung lässt sich natürlich mit sämtlichen Kontrollstrukturen durchführen.

1.7.10 break

Diese Anweisung darf nur innerhalb von `for-`, `foreach while-`, `do..while-` oder `switch-`Anweisungen oder in einem Anweisungsblock, der mit einer bestimmten `case-`Bedingung in einer `switch`-Anweisung verknüpft ist, verwendet werden. Wird die `break`-Anweisung ausgeführt, wird die aktuell ausgeführte Schleife verlassen. Diese Anweisung wird normalerweise dazu verwendet, eine Schleife vorzeitig zu beenden.

Beispiel:

```
<?php
// while-Anweisung (mit Break)
$zufall = 1;
// Ausgabe - 1 2 3 4 5
while ($zufall <= 10) {
        echo $zufall;
        if ($zufall == 5) {
                break;
        }
        $zufall++;
}
?>
```

Diese `while`-Schleife wird so lange durchlaufen, bis die Variable `$zufall` den Wert 5 aufweist. Sollte dies der Fall sein, wird der Schleifendurchlauf durch `break` beendet.

1.7.11 continue

Die `continue`-Anweisung steht mit der `break`-Anweisung in einem sehr engen Zusammenhang. Wie schon die `break`-Anweisung, kann auch `continue` nur innerhalb von `while`-, `do-while-`, `for`- und `foreach`-Anweisungen verwendet werden. Wenn die `continue`-Anweisung ausgeführt wird, wird der aktuelle Durchlauf der ausgeführten Schleife beendet und die nächste Iteration begonnen. Die `continue`-Anweisung verhält sich dabei in jeder Schleifenart unterschiedlich.

- In einer `while`-Schleife weist `continue` den Interpreter an, den restlichen Teil der Schleife zu übergehen und an den Anfang der Schleife zu springen, wo die Bedingung geprüft wird.
- In einer `do-while`-Schleife weist `continue` den Interpreter an, den restlichen Teil der Schleife zu übergehen und an das Ende der Schleife zu springen, wo die Bedingung geprüft wird.
- In einer `for`-Schleife weist `continue` den Interpreter an, den restlichen Teil der Schleife zu übergehen und zur Auswertung des auf die `for`-Schleife folgenden Ausdrucks zu springen.
- In einer `foreach`-Schleife weist `continue` den Interpreter an, den restlichen Teil der Schleife zu übergehen und zurück an den Anfang der Schleife zu springen, wo der nächste Wert in der Aufzählung verarbeitet wird.

Beispiel:

```
<?php
// while-Anweisung (mit continue)
$zufall = 1;
// Ausgabe - 1 2 3 4 5 6 7 8 9 10
while ($zufall <= 10) {
        echo $zufall;
        $zufall++;
        continue;
        echo "Ich werde nie aufgerufen";
}
?>
```

Beispiel:

```
<?php
$zaehler = 0;
$max = 10;
while($zaehler < $max) {
        if ($zaehler % 2) {
                $zaehler++;
                continue;
        }
        echo "Zähler: $zaehler <br>";
        $zaehler++;
}
?>
```

Ausgabe:

```
Zähler: 0
Zähler: 2
Zähler: 4
Zähler: 6
Zähler: 8
```

Hinweis: `continue` kann optional ein numerisches Argument erhalten, das angibt, wie viele Ebenen von enthaltenen Schleifen übersprungen werden sollen.

1.8 Funktionen und Prozeduren

Funktionen dienen dem Zusammenfassen mehrerer Befehle zu einem Aufruf. Dadurch werden Programme lesbarer, weil klar ist, wozu ein Befehlsblock dient.

Bei einigen Programmiersprachen findet eine Unterscheidung zwischen Funktionen statt, die einen Wert zurückgeben, und solchen, die keinen Wert zurückgeben. In Pascal/Delphi etwa gibt es neben Funktionen, die einen Wert zurückgeben, die Prozeduren, die keinen Wert zurückgeben. PHP macht hier, genau wie C und C++, keinen Unterschied.

Definition:

```
FUNCTION MEINEFUNK($ARG_1, $ARG_2, ..., $ARG_N) {
ANWEISUNG/EN;
RETURN $RETVAL;
}
```

Die Funktion erhält die Argumente *'Arg 1'* bis *'Arg n'* übergeben und gibt den Wert der Variablen *'retval'* zurück. Wird kein *'return'* in der Funktion benutzt, hat man dasselbe Verhalten wie bei einer Prozedur in Pascal/Delphi. Rückgabewerte müssen, im Gegensatz zu Pascal/Delphi, nicht abgefragt werden.

Beispiel:

```
function quadratSumme($num) {
       return $num * $num ;
}
echo quadratSumme(4); // Ergebnis: 16
```

return

Der Befehl `return` enthält als Parameter den Rückgabewert. Dies kann ein Ausdruck oder eine einzelne Variable sein. An welcher Stelle innerhalb der Funktion Sie `return` einsetzen, spielt keine Rolle. Auch die mehrfache Notation ist zulässig – hier wird nach dem Motto »Wer zuerst kommt, malt zu erst« verfahren, und die Funktion wird sofort verlassen. Aus Gründen sauberer Programmierung sollten Sie jedoch `return` nur einmal an Ende einer Funktion einsetzen.

Späte Bindung in PHP

Seit PHP4 können Sie eine Funktion an jeder beliebigen Stelle Ihres Skripts definieren. Der PHP-Interpreter verarbeitet als Erstes sämtliche Funktionsdefinitionen und anschließend den anderen Bestandteile des Skripts.

1.8.1 Funktionsargumente

Die Übergabe von Argumenten an die Funktion, die dann dort Parametern entsprechen, kann auf unterschiedliche Art und Weise erfolgen. Im einfachsten Fall geben Sie Variablennamen an:

```
function schreiben($spruch, $vorname) {
 echo "$spruch, $vorname";
}
```

Der Aufruf der Funktion kann nun erfolgen, indem Werte eingesetzt werden:

```
schreiben("Herzlich Willkommen", "Caroline");
```

Der Rückgabewert interessiert hier nicht, also wird auch kein `return` benötigt. Beim Aufruf können natürlich auch Variablen eingesetzt werden. Das folgende Beispiel entspricht der ersten Variante:

```
$vorname = "Caroline";
$spruch = "Herzlich Willkommen";

function schreiben($spruch, $vorname) {
 echo "$spruch, $vorname";
}
schreiben($spruch, $vorname);
```

Lokale und globale Variablen

Die Variablennamen können gleich sein, da Variablen innerhalb einer Funktion lokal sind und nicht im Konflikt mit den globalen Variablen stehen.

Beispiel:

```
<?php
function schreiben($spruch, $vorname) {
 echo "$spruch, $vorname <br>";
 $spruch = "Neuer Spruch";
 $vorname = "Neuer Name";
}
$vorname = "Caroline";
$spruch = "Herzlich Willkommen";
schreiben($spruch, $vorname);
schreiben($spruch, $vorname);
?>
```

Ausgabe:

```
Herzlich Willkommen, Caroline
Herzlich Willkommen, Caroline
```

Übergabe per Referenz

Die Veränderung der übergebenen Variablen kann aber manchmal erwünscht sein. So könnte eine Funktion sämtliche übergebenen Werte in `$inhalt` kursiv schreiben, beispielsweise für Zitate.

```
<?php
function schreibeZitat(&$inhalt) {
 $inhalt = "<i>$inhalt</i>";
}
$spruch = "Hallo Welt!";
echo "$spruch<br>";
schreibeZitat($spruch);
echo $spruch;
?>
```

Ausgabe:

```
Hallo Welt!
Hallo Welt!
```

Der Name der Variablen spielt keine Rolle. Entscheidend ist die Kennzeichnung der Argumente mit &. In diesem Fall wird der globalen Variablen der neue Wert zugewiesen. Diese Vorgehensweise wird als Parameterübergabe durch Referenz bezeichnet (by reference). Der normale Weg ist die Übergabe von Werten (by value), Änderungen wirken sich dann nicht auf die Quelle aus.

1.8.2 Vorgabewerte für Parameter

Eine Funktion kann C++-artige Vorgabewerte für skalare Parameter wie folgt definieren:

```
<?php
function mixen ($typ = "Kaffee") {
    return "Tasse $typ<br>";
}
echo mixen ();
echo mixen ("Tee");
?>
```

Ausgabe:

```
Tasse Kaffee
Tasse Tee
```

Der Vorgabewert muss ein konstanter Ausdruck sein, darf also keine Variable und kein Element einer Klasse sein. Bitte beachten Sie, dass alle Vorgabewerte rechts von den Nicht-Vorgabeparametern stehen sollten – sonst wird es nicht funktionieren.

Beispiel:

```
<?php
function mixen ($typ = "Maxi", $geschmack) {
    return " $typ Becher $geschmack-Mix.";
}
echo mixen ("Kirsch");
?>
```

Ausgabe:

```
Warning: Missing argument 2 for mixen() in C:\php5xampp-
dev\htdocs\php5\test2.php on line 2
Kirsch Becher -Mix.
```

Lösung:

```
<?php
function mixen ($geschmack, $typ = "Maxi") {
    return "$typ Becher $geschmack-Mix.";
}
echo mixen ("Kirsch");
?>
```

Ausgabe:

```
Maxi Becher Kirsch-Mix.
```

1.8.3 Variable Argumentlisten

In PHP3 konnten noch keine variablen Argumentlisten verwendet werden. Um diesen Nachteil zu umgehen, wurden Arrays eingesetzt. Sie können die Anzahl der Argumente dann mit Hilfe von Array-Funktionen bestimmen.

Beispiel:

```
<?php
function formatieren($tag,$argumente) {
    $anzahlargs = count($argumente);
    for ($i = 0; $i < $anzahlargs; $i++) {
      $resultat .= "<".$tag.">".$argumente[$i]."</".$tag.">";
    }
    return $resultat;
}
$personen = array("Matthias","Caroline","Gülten");
// Ausgabe - Kursiv
echo formatieren ("i",$personen) . "<br>";
// Ausgabe - Unterstrichen
echo formatieren ("u",$personen);
?>
```

Ausgabe:

MatthiasCarolineGülten
MatthiasCarolineGülten

Wie man sieht, besteht der Trick darin, `array` zu verwenden. Die Anzahl der Argumente kann leicht mit der Funktion `count` ermittelt werden. Für die Übergabe wird das Array mit der Funktion `array` aus Einzelwerten erzeugt. Wie viele dies sind, spielt nun keine Rolle mehr.

Seit PHP4 sind variable Argumentlisten zulässig. Dies wird bei internen Funktionen wie beispielsweise `max` verwendet. Die Funktion ermittelt den maximalen Wert einer beliebig langen Liste von Argumenten.

Es gibt spezielle Funktionen, mit denen der Parameterblock untersucht werden kann:

- `func_num_args` – Diese Funktion gibt die Anzahl der Parameter zurück.
- `func_get_arg` – Hiermit ermitteln Sie einen konkreten Parameter, die Auswahl erfolgt durch eine Nummer.
- `func_get_args` – Diese Funktion gibt ein Array mit den Parametern zurück.

Beispiele:

```
<?php
function meinefunk() {
    $anzahlargs = func_num_args();
    echo "Anzahl der Argumente: $anzahlargs";
}
// Ausgabe - Anzahl der Parameter (3)
meinefunk (10, 20, 30);
?>
```

```
<?php
function meinefunk() {
    $anzahlargs = func_num_args();
    echo "Anzahl der Argumente: $anzahlargs<br>";
    if ($anzahlargs >= 2) {
        echo "Das 2. Argument ist: " . func_get_arg (1);
    }
}
// Ausgabe
// Anzahl der Argumente: 3
// Das 2. Argument ist: 20
meinefunk (10, 20, 30);
?>
```

```
<?php
function meinefunk() {
    $anzahlargs = func_num_args();
    $arg_liste = func_get_args();
    for ($i = 0; $i < $anzahlargs; $i++) {
        echo "Argument $i ist: " . $arg_liste[$i] . "<br>";
    }
}
```

```
// Ausgabe
// Argument 0 ist: 10
// Argument 1 ist: 20
// Argument 2 ist: 30
meinefunk (10, 20, 30);
?>
```

1.8.4 Rückgabewerte

Wie Sie bereits erfahren haben, können Sie mit dem optionalen Befehl `return` Werte zurückgeben. Es können Variablen jedes Typs zurückgegeben werden, auch Listen oder Objekte. Sie beenden sofort die Funktion, und die Kontrolle wird wieder an die aufrufende Zeile zurückgegeben.

Rückgabe mehrere Werte

Ist es möglich, mehr als einen Wert zurückzugeben? Nein. Ein ähnliches Resultat kann man jedoch durch die Rückgabe von Listen erreichen. Hilfestellung gibt dabei der Befehl `list`.

Beispiel:

```
<?php
function ausgaben() {
   return array (10, 20, 30);
}
list ($erste, $zweite, $dritte) = ausgaben();
// Ausgabe (20)
echo $zweite;
?>
```

1.8.5 Fehlercode als Rückgabewert

Eine komplexe Funktion sollte nicht nur den üblichen Rückgabewert erzeugen, sondern bei Bedarf auch noch einen Fehlercode. Dieser Fehlercode wird ebenfalls in einer `return`-Anweisung definiert. Kann die Funktion die erwartete Operation nicht ausführen, soll sie stattdessen den Fehlercode liefern. Der Entwickler ist dann in der Lage, den Rückgabewert in einer `if`-Abfrage auszuwerten und für die Programmsteuerung zu nutzen. Das folgende Beispiel erfüllt diese Anforderung:

Beispiel:

```
function bruttoberechnen($betrag, $mwst) {
        if ($betrag > 0 && $mwst > 0) {
                return $betrag + ($betrag * $mwst / 100);
        } else {
                return -1;
        }
}
```

Die Funktion erwartet die Angabe eines Nettobetrags und der Mehrwertsteuer, daraus wird dann der Bruttobetrag errechnet. Es wird überprüft, ob die Variablen einen Wert größer 0 liefern. Nur wenn das der Fall ist, erfolgen Berechnung und Rückgabe des berechneten Wertes. Wenn eine der Variablen nur einen Wert kleiner oder gleich 0 enthält, wird der alternative `else`-Zweig ausgeführt. Die dort untergebrachte `return`-Anweisung erzeugt dann den Rückgabewert –1. Diesen Wert verwenden Sie als Fehlercode. Beim Aufruf der Funktion können Sie den Fehlercode berücksichtigen und auswerten.

```
$resultat = bruttoberechnen (100, 0);        // -1

if ($resultat > -1) {
       echo $resultat;
} else {
       echo "Falsche Argumente!";
}
```

Hinweis: In den vorangegangenen Beispielen wurde als Rückgabewert für die Anzeige einer Fehlerfunktion immer der Wert –1 verwendet. Sie können natürlich jeden beliebigen Wert dafür vorgeben. Der Wert sollte sich nur deutlich von den normalen Funktionswerten unterscheiden. Viele Funktionen geben bei einem Fehler beispielsweise den Wahrheitswerts `false` (0) zurück. Bei String-Funktionen können Sie auch einen Leerstring verwenden.

1.8.6 Dynamisch Funktionen erzeugen

Sie möchten eine Funktion anlegen und definieren, während das Skript vom PHP-Interpreter abgearbeitet wird. Hierfür stellt Ihnen PHP die Funktion `create_function()` zur Verfügung.

Beispiel:

```
<?php
$addieren = create_function('$a,$b', 'return $a+$b;');
// Ausgabe (15)
echo $addieren(10,5)
?>
```

Der erste Parameter für `create_function()` ist ein String, der die Argumente der Funktion enthält, und der zweite ist der Anweisungsblock. Die Verwendung von `create_function()` ist außerordentlich langsam. Sie sollten daher eine Funktion vorab definieren und nur in Ausnahmefällen auf `create_function()` zurückgreifen.

1.8.7 Bedingte Funktionen

Wenn eine Funktion nur unter bestimmten Bedingungen definiert wird, muss die Definition dieser Funktion noch vor deren Aufruf abgearbeitet werden.

```
<?php
$signal = TRUE;
```

```
function meinefunk() {
  echo "Wurde aufgerufen!";
}
// Ausgabe - Wurde aufgerufen
if ($signal) meinefunk();
?>
```

1.8.8 Verschachtelte Funktionen

Sie können in PHP Kontrollstrukturen verschachteln, dies gilt auch für Funktionen. Im Folgenden Beispiel wird eine Funktion `zeigeAutoren()` definiert. Sie enthält zwei weitere Funktionen `zeigeAutor()` und `zeigeAutorin()`.

Beispiel:

```
<?php
// Verschachtelte Funktionen
function zeigeAutoren() {
       function zeigeAutor() {
              echo "Matthias Kannengiesser";
       }
       function zeigeAutorin() {
              echo "Caroline Kannengiesser";
       }
       zeigeAutor();
       zeigeAutorin();
}
// Aufruf
zeigeAutoren();
?>
```

Ausgabe:

```
Matthias Kannengiesser
Caroline Kannengiesser
```

Sie greifen, wie Sie sehen, auf die verschachtelten Funktion zu, indem Sie die übergeordnete Funktion `zeigeAutoren()` aufrufen. Dabei verhalten sich die verschachtelten Funktionen wie lokale Variablen. Sie sind somit nur für die übergeordnete Funktion zugänglich (Parent-Funktion). Folgender Aufruf wäre daher außerhalb der Parent-Funktion nicht möglich:

```
// Aufruf (nicht möglich)
zeigeAutor();
```

Tipp: Mit Hilfe der verschachtelten Funktionen haben Sie die Möglichkeit, den Zugriff von außen auszuschließen.

1.8.9 Variablenfunktionen

PHP unterstützt das Konzept der Variablenfunktionen. Wenn Sie an das Ende einer Variablen Klammern hängen, versucht PHP eine Funktion aufzurufen, deren Name der aktuelle Wert der Variablen ist. Dies kann unter anderem für Callbacks, Funktionstabellen usw. genutzt werden.

Beispiel:

```
<?php
function meinefunk()
{
    echo "In meinefunk()<br>";
}

function meinefunk2($arg = '')
{
    echo "In meinefunk2(); der Parameter ist '$arg'.<br>";
}

$func = 'meinefunk';
$func();        // ruft meinefunk() auf

$func = 'meinefunk2';
$func('Funk: Hallo Welt');  // ruft meinefunk2() auf
?>
```

Ausgabe:

```
In meinefunk()
In meinefunk2(); der Parameter ist 'Funk: Hallo Welt'.
```

Variablenfunktionen funktionieren nicht mit Sprachkonstrukten wie `echo()`, `print()`, `unset()`, `isset()`, `empty()`, `include()` und `require()`. Sie müssen Ihre eigenen Wrapperfunktionen verwenden, um diese Konstrukte als variable Funktionen benutzen zu können.

Beispiel:

```
<?php
// Wrapperfunkiton für echo
function sendeecho($string)
{
    echo $string;
}

$func = 'sendeecho';
$func('Echo: Hallo Welt');  // ruft sendeecho() auf
?>
```

Ausgabe:

```
Echo: Hallo Welt
```

1.8.10 Rekursive Funktionen

Sie werden nun noch eine Methode kennen lernen, Funktionen zu verwenden. Es handelt sich um die rekursive Funktion. Dies ist eine Funktion, die sich selbst aufruft. Rekursive Funktionen werden vor allem dort eingesetzt, wo man nicht genau vorherbestimmen kann, wie verschachtelt eine Datenstruktur ist.

Rekursion allgemein

Unter einer Rekursion versteht man die Definition eines Programms, einer Funktion oder eines Verfahrens durch sich selbst. Rekursive Darstellungen sind im Allgemeinen kürzer und leichter verständlich als andere Darstellungen, da sie die charakteristischen Eingenschaften einer Funktion betonen.

Ein Algorithmus heißt rekursiv, wenn er Abschnitte enthält, die sich selbst aufrufen. Er heißt iterativ, wenn bestimmte Abschnitte des Algorithmus innerhalb einer einzigen Ausführung des Algorithmus mehrfach durchlaufen werden. Iteration und Rekursion können oft alternativ in Programmen eingesetzt werden, da man jede Iteration in eine Rekursion umformen kann, und umgekehrt. In der Praxis liegt jedoch oftmals die iterative oder die rekursive Lösung auf der Hand und die dazu alternative Form ist gar nicht so leicht zu bestimmen.

Hinweis: Programmtechnisch läuft eine Iteration auf eine Schleife, eine Rekursion dagegen auf den Aurfuf einer Methode durch sich selbst hinaus.

Fallbeispiel:

Nehmen Sie einen Papierstreifen und versuchen Sie ihn so zu falten, dass sieben genau gleich große Teile entstehen. Dabei dürfen Sie kein Lineal oder sonstiges Hilfsmittel verwenden. Sie werden feststellen, dass die Aufgabe gar nicht so einfach ist.

Wenn Sie statt sieben jedoch acht Teile machen, wird es plötzlich einfach: Einmal in der Mitte falten, dann nochmals falten .....

Genau das ist das Prinzip der Rekursion: Ein Problem wird auf ein »kleineres« Problem zurückgeführt, das wiederum nach demselben Verfahren bearbeitet wird. Rekursion ist eine wichtige algorithmische Technik.

Am obigen Beispiel haben Sie auch gesehen, dass die Lösung einer Aufgabe, wenn sie mit Rekursion möglich ist, sehr einfach gelöst werden kann. Hier nun zwei rekursive Fallbeispiele.

Fakultät einer Zahl n (n!) rekursiv

Bei der Berechnung der Fakultätsfunktion geht man aus von der Definition der Fakultät:

```
0! = 1
n! = 1 * 2 * 3 * ... * n für n>0
```

Man beginnt bei den kleinen Zahlen. Der Wert von *O! ist 1*, der Wert von *1! ist 0!*1*, der Wert von *2! ist 1!*2*, der Wert von *3! ist 2!*3* usw.

Nimmt man eine Schleifenvariable `$i`, die von 1 bis n durchgezählt wird, so muss innerhalb der Schleife lediglich der Wert der Fakultät vom vorhergehenden Schleifendurchlauf mit dem Wert der Schleifenvariablen multipliziert werden.

Lösung 1 (iterativ):

```
<?php
function fak($n) {
       $resultat = 1;
    for ($i=1; $i<=$n; $i++) {
       $resultat = $i*$resultat;
       }
    return $resultat;
}
echo fak(1) . "<br>";
echo fak(2) . "<br>";
echo fak(3) . "<br>";
echo fak(4) . "<br>";
?>
```

Ausgabe:

```
1
2
6
24
```

Bei der rekursiven Berechnung der Fakultätsfunktion geht man ebenfalls von der Definition der Fakultät aus, beginnt jedoch nicht bei den kleinen Zahlen, sondern bei den großen Zahlen und läuft dann zu den kleinen Zahlen zurück (recurrere = lat. zurücklaufen).

```
n! = 1 * 2 * 3 * ... * n für n>0
0! = 1
```

Im Gegensatz zur Iteration schaut man jetzt auf die Funktion *f(n)* und versucht, diese Funktion durch sich selbst, aber mit anderen Aufrufparametern darzustellen. Die mathematische Analyse ist hier ziemlich leicht, denn man sieht sofort, dass

*f(n) = n * f(n-1)*

ist. Damit hat man das Rekursionsprinzip bereits gefunden. Die Rekursion darf jedoch nicht ewig andauern, sie muss durch ein Abbruchkriterium angehalten werden. Dies ist die Bedingung *0!=1*.

Lösung 2 (rekursiv):

```
<?php
function fak($n){
     if ($n==0) {
          return 1;
```

```
        } else {
         return $n*fak($n-1);
        }
}

echo fak(1) . "<br>";
echo fak(2) . "<br>";
echo fak(3) . "<br>";
echo fak(4) . "<br>";
?>
```

Ausgabe:

```
1
2
6
24
```

Der `else`-Zweig wird angesprungen, wenn die Abbruchbedingung nicht erreicht wird. Hier ruft die Methode sich selbst wieder auf. Dabei ist zu beachten, dass die Anweisung, die die Methode aufruft, noch gar nicht abgearbeitet werden kann, solange die aufgerufene Methode kein Ergebnis zurückliefert.

Der `if`-Zweig wird angesprungen, wenn die Abbruchbedingung erreicht ist.

Um Ihnen die Analyse zu vereinfachen, haben wir die rekursive Lösung etwas angepasst.

```
<?php
function fak($n){
       //Aufruf
       echo "Eintritt mit $n<br>";
              if ($n==0) {
                     return 1;
              } else {
                     $ergebnis = $n*fak($n-1);
                     // Rücksprung
                     echo "Austritt mit $n: $ergebnis<br>";
                     return $ergebnis;
              }
}

fak(4);
?>
```

Ausgabe:

```
Eintritt mit 4
Eintritt mit 3
Eintritt mit 2
Eintritt mit 1
Eintritt mit 0
Austritt mit 1: 1
Austritt mit 2: 2
Austritt mit 3: 6
Austritt mit 4: 24
```

Zu jedem Aufruf gehört auch genau ein Rücksprung! Sie können dies beim Programmablauf mit Hilfe der eingefügten Ausgabezeilen nachvollziehen.

Man beachte die Anzahl der Aufrufe. Im iterativen Fall wird die Methode ein einziges Mal aufgerufen und im Schleifenkörper n mal durchlaufen. Bei der rekursiven Berechnung wird die Methode n+1 mal aufgerufen. Dabei muss jedes Mal Speicherplatz auf dem Stack reserviert werden. Da Parameter als lokale Variablen kopiert werden, wird auch dabei Speicherplatz verbraucht. Bei Rekursionen ist daher unbedingt darauf zu achten, dass die Abbruchbedingung bzw. das Rekursionsende korrekt implementiert wurde.

Türme von Hanoi

Ein Turm aus n verschieden großen Scheiben soll mit möglichst wenig Zügen (Umsetzungen) vom Startplatz S auf den Zielplatz Z transportiert werden. Ein dritter Platz steht als Hilfsplatz H zur Verfügung. Dabei gelten die folgenden Spielregeln:

- Jeder Zug besteht darin, eine Scheibe zu bewegen.
- Niemals darf eine größere Schiebe über einer kleineren Scheibe zu liegen kommen.

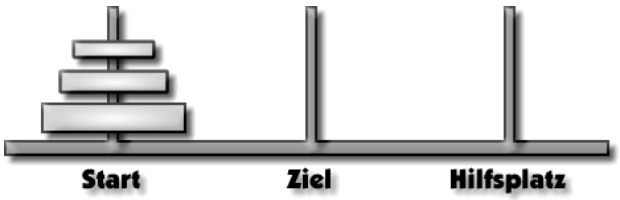

Bild 1.9: Türme von Hanoi

Schlüsselprinzip: Rekursion

Wenn wir das Problem in einem etwas einfacher gelagerten Fall lösen können, dann kann man diese Lösung auch für den schwierigeren Fall verwenden.

2 Scheiben:

- - übertrage den Turm mit 1 Scheibe vom Start- auf den Hilfsplatz
- - bewege die Scheibe 2 vom Start- auf den Zielplatz
- - übertrage den Turm mit 1 Scheibe vom Hilfs- auf den Zielplatz

3 Scheiben:

- - übertrage den Turm mit 2 Scheiben vom Start- auf den Hilfsplatz
- - bewege die Scheibe 3 vom Start- auf den Zielplatz
- - übertrage den Turm mit 2 Scheiben vom Hilfs- auf den Zielplatz
- ...

n Scheiben:

- - übertrage den Turm mit n-1 Scheiben vom Start- auf den Hilfsplatz
- - bewege die Scheibe n vom Start- auf den Zielplatz
- - übertrage den Turm mit n-1 Scheiben vom Hilfs- auf den Zielplatz

Syntax der Aufrufe: (beachten Sie die Baumstruktur)

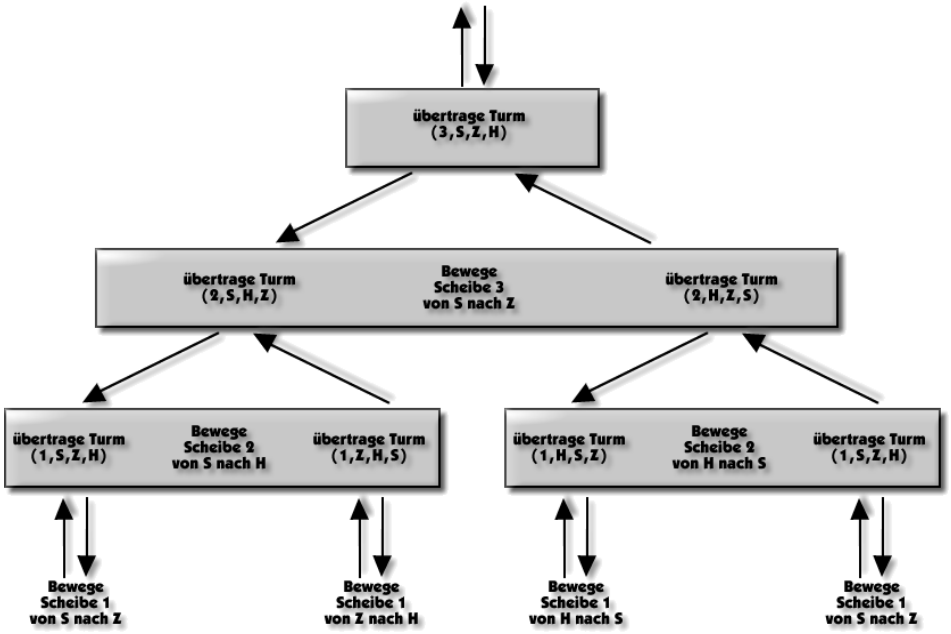

Bild 1.10: Ablauf der Rekursion

Lösung:

```
<?php
function setzeTurm($n, $start, $ziel, $hilf) {
    if ($n>0)  {
                setzeTurm ($n-1, $start, $hilf, $ziel);
                echo("Bewege Scheibe $n vom $start-Platz zum $ziel-
Platz.<br>");
                setzeTurm ($n-1, $hilf, $ziel, $start);
    }
}

setzeTurm (3,'Start','Ziel','Hilfsplatz');
?>
```

Ausgabe:

```
Bewege Scheibe 1 vom Start-Platz zum Ziel-Platz.
Bewege Scheibe 2 vom Start-Platz zum Hilfsplatz-Platz.
```

```
Bewege Scheibe 1 vom Ziel-Platz zum Hilfsplatz-Platz.
Bewege Scheibe 3 vom Start-Platz zum Ziel-Platz.
Bewege Scheibe 1 vom Hilfsplatz-Platz zum Start-Platz.
Bewege Scheibe 2 vom Hilfsplatz-Platz zum Ziel-Platz.
Bewege Scheibe 1 vom Start-Platz zum Ziel-Platz.
```

Weitere Beispiele für rekursive Probleme sind:

- Wege aus einem Labyrinth
- Sortierverfahren
- Szierpinski-Dreiecke
- Baum des Pythagoras
- Kockkurven
- Julia- und Mandelbrotmengen
- Logistisches Wachstum
- Fibonacchi-Folge
- Springer-Problem
- 8-Damen-Problem

1.9 Referenzen in PHP

1.9.1 Was sind Referenzen?

Referenzen sind in PHP ein Mechanismus, um verschiedene Namen für den gleichen Inhalt von Variablen zu ermöglichen. Sie sind nicht mit Zeigern in C zu vergleichen, sondern Aliasdefinitionen für die Symboltabelle. PHP unterscheidet zwischen:

- Variablenname
- Variableninhalt

Der gleiche Variableninhalt kann unterschiedliche Namen besitzen. Der bestmögliche Vergleich ist der mit Dateinamen und Dateien im Dateisystem von Unix:

- Variablennamen sind Verzeichniseinträge.
- Variableninhalt ist die eigentliche Datei.

Referenzen können als Hardlinks im Dateisystem verstanden werden.

1.9.2 Was leisten Referenzen?

PHP-Referenzen erlauben es, zwei Variablennamen sich auf den gleichen Variableninhalt beziehen zu lassen. Das heißt im folgenden Beispiel, dass sich `$punkte` und `$punktestand` auf dieselbe Variable beziehen:

```
<?php
$punkte = 1000;
$punktestand = &$punkte;
```

```
// Ausgabe (1000)
echo $punktestand;
?>
```

Achtung: `$punkte` und `$punktestand` sind hier gleichwertig, und `$punkte` ist nicht nur ein Zeiger auf `$punktestand` oder umgekehrt, sondern `$punkte` und `$punktestand` zeigen auf denselben Inhalt.

Seit PHP 4.0.4 kann `&` auch in Verbindung mit `new` verwendet werden.

```
<?php
class Haus
{
        var $etagen;
        function Haus($etagen)
        {
                $this->etagen = $etagen;
        }
}
$meinhaus = &new Haus(2);
$hausetagen = &$meinhaus->etagen;
// Ausgabe (2)
echo $hausetagen;
$hausetagen = 10;
// Ausgabe (10)
echo $meinhaus->etagen;
?>
```

Wenn der `&`-Operator nicht verwendet wird, erzeugt PHP eine Kopie des Objekts. Wenn nun `$this` innerhalb der Klasse verwendet wird, bezieht es sich auf die aktuelle Instanz der Klasse. Die Zuordnung ohne `&` erzeugt eine Kopie der Instanz (d.h. des Objekts) und `$this` wird sich auf die Kopie beziehen. In der Regel will man aus Performance- und Speicherverbrauchsgründen nur eine einzige Instanz einer Klasse erzeugen.

pass-by-reference

Eine weitere Einsatzmöglichkeit von Referenzen ist die Übergabe von Parametern an eine Funktion mit *pass-by-reference*. Dabei beziehen sich der lokale Variablenname und auch der Variablenname der aufrufenden Instanz auf denselben Variableninhalt:

```
<?php
function ausgabe(&$var) {
    return $var++;
}

$zahl=5;
// Ausgabe (5)
echo ausgabe ($zahl);
// Ausgabe (6)
echo $zahl;
?>
```

return-by-reference

Daneben besteht die Möglichkeit, aus Funktionen heraus Werte mit *return-by-reference* zurückzugeben. Das Zurückgeben von Ergebnissen per Referenz aus Funktionen heraus kann in manchen Fällen recht nüzlich sein. Dabei ist folgende Syntax zu beachten:

```
<?php
function &ausgabe($param) {
    return $param;
}

$wert =&ausgabe(5);
// Ausgabe (5)
echo $wert;
?>
```

In diesem Beispiel wird also die Eigenschaft des von `ausgabe()` gelieferten Wertes gesetzt, nicht die der Kopie, wie es der Fall wäre, wenn die Funktion `ausgabe()` ihr Ergebnis nicht per Referenz liefern würde.

Achtung: Im Gegensatz zur Parameterübergabe per Referenz ist bei der Rückgabe mittels Referenz an beiden Stellen die Angabe des & notwendig.

1.9.3 Referenzen aufheben

Wird eine Referenz aufgehoben, so wird nur die Bindung zwischen einem Variablennamen und dem Variableninhalt entfernt. Der Inhalt der Variablen wird nicht gelöscht.

```
<?php
$wert = 10;
$zahl =&$wert;
unset ($wert);
// Ausgabe (10)
echo $zahl;
?>
```

Die Variable `$zahl` wird nicht gelöscht, sondern es wird nur die Bindung des Variablennamens `$wert` an den Variableninhalt aufgehoben. Dieser Variableninhalt ist immer noch über `$zahl` verfügbar.

Hinweis: Auch in diesem Fall sieht man die Analogie zwischen Unix und den Referenzen: Das Aufheben einer Referenz entspricht einem Aufruf von `unlink` unter Unix.

1.9.4 Referenzen entdecken

Viele Sprachelemente von PHP sind intern mit der Benutzung von Referenzen implementiert, daher gilt alles bisher Gesagte auch für folgende Konstrukte:

global references

Die Verwendung von `global $var` erzeugt im aktuellen Gültigkeitsbereich eine Referenz auf die globale Variable `$var`, sie ist also äquivalent zu Folgendem:

```
$var = &$GLOBALS["var"];
```

Dies hat zur Folge, dass das Anwenden von `unset()` auf `$var` keinen Einfluss auf die globale Variable hat.

$this

In einer Objektmethode ist `$this` immer eine Referenz auf die aufrufende Objektinstanz.