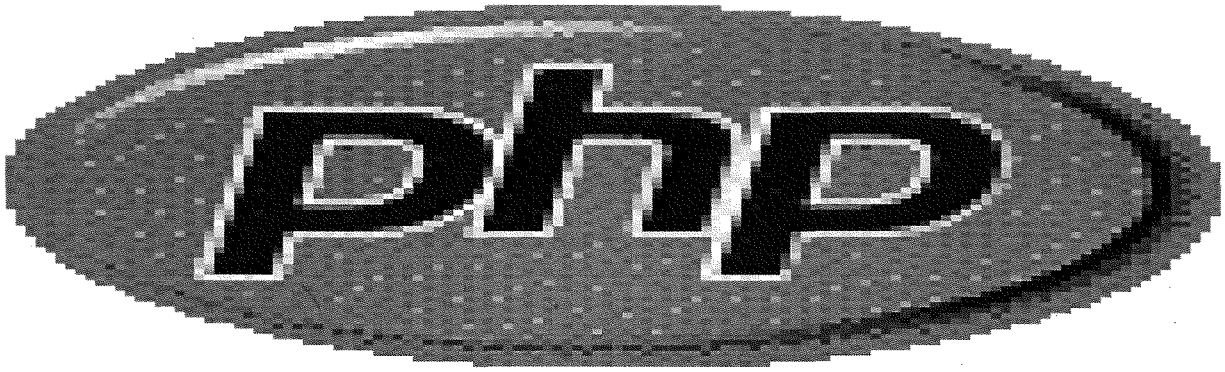

Inhalt:

1. Einführung

PHP ist die Abkürzung für "PHP: Hypertext Preprocessor" und ist eine serverseitige Skriptsprache die sich in HTML einbinden lässt. Viele der syntaktischen Möglichkeiten sind den Programmiersprachen C, Java und Perl entnommen und es wurden auch einige PHP spezifische Features entwickelt. Das Ziel der Sprache ist es, das Schreiben von Programmen zur Erzeugung von dynamisch generierten Seiten zu erleichtern und zu beschleunigen.

Ein einleitendes Beispiel

```
<html>
 <head>
  <title>Beispiel</title>
 </head>
 <body>
  <?php echo "Hallo, ich bin ein PHP-Skript!"; ?>
 </body>
</html>
```

Dieses Skript unterscheidet sich von einem CGI-Skript, das in einer Sprache wie Perl oder C geschrieben wurde -- anstatt ein Programm mit vielen Anweisungen zur Ausgabe von HTML zu schreiben, schreibt man einen HTML Code mit einigen, eingebetteten Anweisungen, um etwas auszufügen (z.B. um -wie oben- Text auszugeben). Der PHP Code steht zwischen speziellen Anfangs- und Schlusstags, mit denen man in den PHP-Modus und zurück wechseln kann. Was PHP von client-seitigen Sprache wie Javaskript unterscheidet ist, dass der Code vom Server ausgeführt wird.

Sollten sie einen Skript wie den obigen auf ihrem Server ausführen, würde der Besucher nur das Ergebnis dessen empfangen, ohne die Möglichkeit zu haben, herauszufinden, wie der zugrundeliegende Code aussieht.

2. Typen und Variablen

Der Typ einer Variabalen wird normalerweise nicht vom Programmierer bestimmt; vielmehr wird dies zur Laufzeit von PHP entschieden, abhängig vom Zusammenhang in dem die Variable benutzt wird. Wenn sie die Umwandlung in einen bestimmten Variablen-Typ erzwingen wollen, können sie dies entweder per cast oder durch Gebrauch der Funktion settype.
Beachten Sie, dass eine Variable je nach Gebrauch und Situation auf unterschiedliche Art und Weise typisiert sein kann.

2.1 Typen-Tricks

PHP erfordert (bzw. unterstützt) keine explizite Typ-Definitionen bei der Deklaration von Variablen; der Typ einer Variablen wird bestimmt durch den Zusammenhang in dem die Variable benutzt wird. Das bedeutet, dass bei der Zuweisung einer Zeichenkette / eines Strings zu einer Variablen *var* diese Variable *var* den Typ String erhält. Sollten sie dananch der Variablen *var* einen Integer-Wert zuweisen, wird sie zu einem Integer-Typ.

Ein Beispiel für die automatische Typ-Konvertierung von PHP ist der Plus-Operator '+'. Ist einer der zu addierenden Werte vom Typ double, werden alle Werte als double-Typ gehandhabt. Auch das Ergebnis der Addition wird vom Typ double sein. Ist dies nicht der Fall, werden Werte als Integer-Typen angesehen und das Ergebnis wird ebenfalls vom Typ Integer sein. Beachten sie, dass hierdurch nicht der Typ der Additions-Elemente selbst beeinflusst wird; der Unterschied liegt einzig und allein in der Auswertung dieser Elemente.

```
$foo = "0";                      // $foo ist vom Typ String (ASCII 48)
$foo++;                          // $foo ist immer noch vom Typ String, Inhalt "1" (ASCII 49)
$foo += 1;                       // $foo ist jetzt vom Typ Integer (2)
$foo = $foo + 1.3;               // $foo ist nun vom Typ double (3.3)
$foo = 5 + "10 Little Piggies";  // $foo ist vom Typ Integer (15)
$foo = 5 + "10 Small Pigs";      // $foo ist vom Typ Integer (15)
```

2.2 Typ-Umwandlung

Typ-Umwandlung in PHP funktioniert vielfach wie in C: Der Name des geforderten Typs wird vor der umzuwandelnden Variablen in Klammern gesetzt.

```
$foo = 10;   // $foo ist ein Integer-Wert
$bar = (double) $foo;   // $bar ist vom Typ double
```

Folgende Umwandlungen sind möglich:

- (int), (integer) - Umwandlung in Integer-Wert
- (real), (double), (float) - hin zu double
- (string) - hin zu String
- (array) - hin zum Array
- (object) - Wandlung zum Objekt

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

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Es ist nicht immer offenkundig, was bei der Typ-Umwandlung geschieht. Zum besseren Verständnis sollte das Folgende beachtet werden:

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

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0];                    // gibt 'ciao' aus
```

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

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar;  // gibt 'ciao' aus
```

3. Strings / Zeichenketten

Strings können durch eines von zwei der folgenden Sets von Begrenzungszeichen zugewiesen werden:
Wenn der String eingeschlossen wird von doppelten Anführungszeichen ("), werden die Variablen innerhalb der Zeichenkette ausgewertet (begrenzt durch einige Einschränkungen beim "Parsen" (Ausführen) des PHP- Scripts).
Wie in C und Perl kann das Backslash-Zeichen ("\") zur An- / Ausgabe spezieller Zeichen benutzt werden:

Zeichenfolge	Bedeutung
\n	neue Zeile
\r	Wagenrücklauf / an Zeilenanfang
\t	horizontaler Tabulator
\\	Backslash / Rückstrich
\$	Dollar-Symbol
\"	doppelte Anführungszeichen
\[0-7]{1,3}	die Zeichenfolge, die dem regulären Ausdruck entspricht, ist in Oktal-Schreibweise
\x[0-9A-Fa-f]{1,2}	die Zeichenfolge, die dem regulären Ausdruck entspricht ist, in Hexadezimal-Schreibweise

Sie können ein beliebiges anderes Zeichen übergehen bzw. von der Auswertung ausschliessen; in der höchsten Warn-Stufe wird aber eine Warnmeldung ausgegeben.

Die zweite Möglichkeit eine(n) Zeichenkette / String zu begrenzen, ist der Gebrauch des einfachen Anführungszeichens ("'"), auch Single-Quote genannt. Hier werden einzig die "\\" und "\'"-Zeichenfolgen von der Auswertung ausgeschlossen. Dies dient der Erleichterung, so dass sie Single-Quotes und Backslashes in einem durch einfache Anführungszeichen begrenzten String benutzen können. Variablen innerhalb von durch Single-Quotes begrenzten Strings werden nicht ausgewertet.

Zeichenketten / Strings können mittels des '.'-Operators miteinander verbunden werden. Beachten sie, dass hier der '+' (Additions)-Operator nicht funktioniert.

Beispiele:

```
<?php
/* Zuweisung eines Strings. */
$str = "Dies ist eine Zeichenkette";

/* Erweiterung dieses Strings. */
$str = $str . " mit etwas zusätzlichem Text";

/* andere Möglichkeit zum Erweitern incl. Anweisung für Neue-Zeile. */
$str .= " und einer neuen Zeile am Ende.\n";

/* Dieser String wird am Ende zu: '<p>Nummer: 9</p>' */
$num = 9;
$str = "<p>Nummer: $num</p>";

/* Dieser wird zu '<p>Nummer: $num</p>' */
$num = 9;
$str = '<p>Nummer: $num</p>';
?>

/* Das erste Zeichen eines Strings */
$str = 'Das ist ein Test.'
$first = $str[0];

/* Das letzte Zeichen eines Strings */
$str = 'Das ist immer noch ein Test.'
$last = $str[strlen($str)-1];
?>
```

3.1 Umwandlung von Zeichenketten / Strings

Sobald ein String als Zahlenwert angesehen wird, wird der resultierende Wert und Typ wie folgt festgelegt:

Der String wird als double angesehen, wenn er eines der Zeichen '.', 'e', oder 'E' enthält. Ansonsten wird er als Integer-Wert interpretiert.

Der Inhalt wird durch den Anfangsteil des Strings vorgegeben. Sofern der String mit numerischen Daten beginnt, wird ein Zahlen-Wert angenommen. Andererseits kann der Wert auch 0 (Null) sein. Gültig sind auch Werte mit einem optionalen Vorzeichen, gefolgt von einer oder mehreren Zahlen (optional mit einem Dezimalpunkt). Wahlweise kann auch ein Exponent angegeben werden. Dieser besteht aus einem 'e' oder 'E', gefolgt von einer oder meheren Zahlen.

Sobald der erste Ausdruck ein String ist, hängt der Typ der Variablen vom zweiten Ausdruck ab.

```
$foo = 1 + "10.5";                  // $foo ist double (11.5)
$foo = 1 + "-1.3e3";                // $foo ist double (-1299)
$foo = 1 + "bob-1.3e3";                     // $foo ist ein Integer-Wert (1)
$foo = 1 + "bob3";                  // $foo ist ein Integer-Wert (1)
$foo = 1 + "10 Small Pigs";                 // $foo ist ein Integer-Wert (11)
$foo = 1 + "10 Little Piggies";             // $foo ist ein Integer-Wert (11)
$foo = "10.0 pigs " + 1;            // $foo ist ein Integer-Wert (11)
$foo = "10.0 pigs " + 1.0;                  // $foo ist double (11)
```

4. Arrays

Arrays reagieren wie indizierte Arrays (Vektoren).

4.1 Eindimensionale Arrays

Sie können ein Array erzeugen, indem sie die list oder array Funktionen benutzen, oder sie können explizit den Wert eines jeden Array-Elements festlegen.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

Sie können ein Array auch erzeugen, indem sie einfach Werte einem Array hinzu fügen. Wenn sie einer Array-Variablen einen Wert weisen, indem sie leere Klammern benutzen, wird der Wert am Ende des Arrays hinzugefügt.

```
$a[] = "hello";        // $a[2] == "hello"
$a[] = "world";        // $a[3] == "world"
```

Arrays können durch Gebrauch der asort, arsort, ksort, rsort, sort, uasort, usort, und uksort-Funktionen sortiert werden (in Abhängigkeit von der gewünschten Sortierung).
Sie können die Anzahl der Elemente eines Arrays mittels der count-Funktion ermitteln.
Durchlaufen können sie ein Array per next und prev-Funktion.

4.2 Mehrdimensionale Arrays

Mehrdimensionale Arrays sind einfach. Für jede Dimension des Arrays fügen sie einen anderen [key]-Wert ans Ende an:

```
$a[1]     = $f;                // Ein-Dimensionales Beispiel
$a["foo"]  = $f;

$a[1][0]    = $f;              // Zwei-Dimensional
$a["foo"][2] = $f;             // sie können numerische und assoziative Indizes verwenden
$a[3]["bar"] = $f;             # oder so

$a["foo"][4]["bar"][0] = $f;   # Vier-Dimensional!
```

In PHP ist es nicht möglich, mehrdimensionale Arrays direkt in Strings zu referenzieren. Deshalb erzeugt auch das nachfolgende Beispiel nicht das gewünschte Ergebnis:

```
$a[3]['bar'] = 'Bob';
echo "Dies funktioniert nicht: $a[3][bar]";
```

In PHP wird das o.a. Beispiel folgendes ausgeben: Dies funktioniert nicht: Array[bar] . Jedoch kann der String-Verbindungs- Operator benutzt werden, dies zu umgehen:

```
$a[3]['bar'] = 'Bob';
echo "Das funktioniert: " . $a[3][bar];
```

In PHP4 kann das Problem umgangen werden, indem die Array-Referenz in geschweiften Klammern eingeschlossen wird:

```
$a[3]['bar'] = 'Bob';
echo "Das funktioniert in PHP4: {$a[3][bar]}";
```

Man kann multi-dimensionale Arrays auf viele Arten füllen, aber der trickreichste Weg geht über das Verständnis, wie der array-Befehl für assoziative Arrays zu benutzen ist. Die folgenden Code-Schnippsel füllen das ein-dimensionale Array über den gleichen Weg:

```
$a["color"]      = "red";
$a["taste"]      = "sweet";
$a["shape"]      = "round";
$a["name"]       = "apple";
$a[3]            = 4;

$a = array(
"color" => "red",
"taste" => "sweet",
"shape" => "round",
"name"  => "apple",
3       => 4
);
```

Die array-Funktion kann eingefügt werden für mehrdimensionale Arrays:

```
<?
$a = array(
  "apple"  => array(
    "color"  => "red",
    "taste"  => "sweet",
    "shape"  => "round"
  ),
  "orange"  => array(
    "color"  => "orange",
    "taste"  => "tart",
    "shape"  => "round" ),
  "banana"  => array(
    "color"  => "yellow",
    "taste"  => "paste-y",
    "shape"  => "banana-shaped"
  )
);

echo $a["apple"]["taste"];    # dies wird "sweet" ausgegeben
?>
```

5. Objekte

5.1 Objekt-Initialisierung

Um ein Objekt zu initialisieren benutzen sie die Angabe new, dadurch wird das Objekt einer Variablen-Instanz zugewiesen.

```
class foo
{
    function do_foo ()
    {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();
```

6. Konstanten

PHP definiert eine Reihe von Konstanten und stellt einen Mechanismus zur Verfügung, mit dem man zusätzliche Konstanten zur Laufzeit definieren kann. Konstanten sind Variablen sehr ähnlich, bis auf die Tatsache, dass sie mit der define-Funktion definiert werden müssen und später nicht mehr mit einem anderen Wert versehen werden können.

Folgende vordefinierten Variablen sind immer verfügbar:

__FILE__	Der Name der Skript-Datei, die gerade geparsed wird. Wird diese Konstante in einer Datei verwendet, die per include oder require eingebunden wurde, liefert sie den Namen der eingebundenen Datei, nicht den der aufrufenden Datei.
__LINE__	Die Nummer der Zeile im laufenden Skript, die gerade geparst wird. Wird diese Konstante in einer Datei benutzt, die per include oder require eingebunden wurde, liefert sie die Zeilennummer innerhalb der eingebundenen Datei.
PHP_VERSION	Ein String, der die Versionsnummer des PHP-Parsers enthält, der gerade verwendet wird; z. B. ´3.0.8-dev´.
PHP_OS	Der Name des Betriebsystems, auf dem der PHP-Parser ausgeführt wird; z. B. ´Linux´.
TRUE	Der Wert ´wahr´.
FALSE	Der Wert ´falsch´.
E_ERROR	Bedeutet einen Fehler, der sich von einem ´parsing error´ unterscheidet. Die Ausführung des Skriptes wird beendet.
E_WARNING	Gibt einen Zustand zurück,durch den PHP weiß, dass etwas nicht in Ordnung ist, das aktuelle Skript aber trotzdem weiter ausführt; dies kann vom Skript selbst aufgefangen werden. Ein Beispiel wäre ein ungültiger regulärer Ausdruck (regexp) in der Funktion ereg.
E_PARSE	Der Parser hat Probleme mit ungültiger Syntax in der Skript-Datei. Die Ausführung des Skriptes wird beendet.
E_NOTICE	Etwas ist aufgetreten, das ein Fehler sein kann oder nicht. Das aktuelle Skript wird weiter ausgeführt. Beispiele hierfür sind ein nicht gequoteter string als Hash-Index oder der Zugriff auf eine Variable, die nicht gesetzt wurde. Die E_*-Konstanten werden typischerweise mit der error_reporting-Funktion benutzt, um das Fehlermeldungs-Niveau festzusetzen. Zusätzliche Konstanten können mithilfe der define-Funktion definiert werden. Zu beachten ist, dass dies Konstanten sind, und keine Makros, wie man sie von C her kennt; nur gültige Skalar-Daten können von einer Konstante vertreten werden.

6.1 Konstanten definieren

```
<?php
define("CONSTANT", "Hallo Welt.");
echo CONSTANT;                                  // gibt  "Hallo Welt." aus.
?>
```

Beispiel: Die Benutzung von __FILE__ und __LINE__

```
<?php
function report_error($file, $line, $message) {
   echo "Ein Fehler ist aufgetreten in $file in Zeile $line: $message.";
}


report_error(__FILE__,__LINE__, "Irgendetwas ist falsch gelaufen!");
?>
```

7 Ausdrücke

Ausdrücke (Expressions) sind die wichtigsten Bausteine von PHP. In PHP ist fast alles, was geschrieben wird, ein Ausdruck. Die einfachste, aber auch zutreffenste Definition für einen Ausdruck ist "alles, was einen Wert hat".

Die grundlegendste Formen von Ausdrücken sind Konstanten und Variablen. Wenn man "$a = 5" schreibt, weist man $a den Ausdruck ´5´ zu. ´5´ hat offensichtlich den Wert 5. Anders ausgedrückt: ´5´ ist ein Ausdruck mit dem Wert 5 (in diesem Fall ist ´5´ eine Integer-Konstante).

8. Operatoren

8.1 Arithmetische Operatoren

+	Addition
-	Subtraktion
++	Inkrementieren (x=x+1, x++, ++x)
--	Dekrementieren (x=x-1, x--, --x)
*	Multiplikation
/	Division
%	Modulo (Rest einer Ganzzahldivision)

Beispiele:
```
x=y+10;
++x-20;
x++*10;
a=10%2;
```

8.2 Vergleichsoperatoren

==	gleich
!=	ungleich
>	größer
>=	größer gleich
<	kleiner
<=	kleiner gleich

Beispiele:
```
if(a==b) printf(...);
x=a!=b;
x=a>=b;
```

8.3 Logische Operatoren

!	Negation
&&	UND-Verknüpfung
||	ODER-Verknüpfung

Besipiele:
```
IF(!a) b=2;
x=a&&b;
if((a==2)||(a==5)) printf(...);
```

Ein Wert ungleich Null bedeutet "true" (wahr), gleich Null bedeutet "false" (falsch).!a ist gleichbedeutend mit a==0.

8.4 Operatoren zur Bit-Manipulation

&	bitweise UND
|	bitweise ODER
^	bitweise Exclusive-Oder-Verknüpfung
<	Bit-Verschiebung nach links
>>	Bit-Verschiebung nach rechts
~	Bilden des Einerkomplements

Beispiele:
```
a=x&2;
x=x|2;
x=5^2;
x=x<<2;
a=~x;
```

8.5 String-Operatoren

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!"; // now $a contains "Hello World!"
```

9 Kontroll-Strukturen

Jedes PHP-Skript besteht aus einer Reihe von Anweisungen. Eine Anweisung kann aus einem Funktions-Aufruf, einer Schleife, einer bedingten Anweisung oder einem Befehl, der nichts macht (eine leere Anweisung), bestehen. Jeder Befehl endet gewöhnlich mit einem Semikolon. Darüber hinaus können Befehle zu einer Anweisungsgruppe zusammengefasst werden, welche durch geschweifte Klammern begrenzt wird. Eine Anweisungsgruppe ist auch eine Anweisung. Die unterschiedlichen Arten von Anweisungen werden in diesem Abschnitt erläutert.

9.1 if

Der if-Befehl ist eine der wichtigsten Möglichkeiten vieler Programmier-Sprachen, PHP eingeschlossen. Er erlaubt die bedingte Ausführung von Programmteilen. PHP beinhaltet eine if-Struktur, die ähnlich der von C ist:

```
if (expr)
  Anweisung
```

Wie im Abschnitt über Expressions / Ausdrücke beschrieben, wird expr auf seinen wirklichen Wertinhalt ausgewertet. Wenn *expr* TRUE entspricht, wird PHP Anweisung ausführen, falls nicht - sie also FALSE ist - wird Anweisung übergangen.
Das folgende Beispiel wird a ist grösser als b anzeigen, wenn *$a* grösser ist als *$b*:

```
if ($a > $b)
  print "a ist grösser als b";
```

Oft werden sie die bedingte Ausführung von mehr als einer Anweisung wollen. Selbstverständlich ist es nicht erforderlich, jede Anweisung mit einer if-Bedingung zu versehen. Statt dessen können sie mehrere Anweisungen in Gruppen zusammenfassen. Z.B. wird der folgende Programm-Code a ist grösser als b anzeigen, wenn *$a* grösser ist als *$b*. Danach wird der Wert von *$a* in *$b* gespeichert:

```
if ($a > $b) {
  print "a ist grösser als b";
  $b = $a;
}
```

If-Anweisungen können ohne Einschränkung innerhalb anderer if-Anweisungen definiert werden. Das ermöglicht ihnen völlige Flexibilität bei der bedingten Ausführung verschiedenster Programmteile.

9.2 else

Häufig ist es erforderlich, eine Anweisung auszuführen, wenn eine bestimmte Bedingung erfüllt ist und eine andere Anweisung, falls sie nicht erfüllt ist. Dafür gibt es else. Else erweitert eine if-Anweisung um die Ausführung von Anweisungen, sobald der Ausdruck der if-Anweisung als FALSE angesehen wird. Der folgende Code wird z.B. a ist grösser als b anzeigen, wenn *$a* grösser ist als *$b*, anderenfalls a ist NICHT grösser als b:

```
if ($a > $b) {
  print "a ist grösser als b";
} else {
  print "a is NICHT grösser als b";
}
```

Die else-Anweisung wird nur ausgeführt, wenn der if-Ausdruck als FALSE ausgewertet wurde und wenn bei vorhandenen elseif-Ausdrücken diese ebenfalls FALSE sind (siehe unten).

9.3 elseif

Elseif ist, wie der Name schon sagt, eine Verbindung von if und else. Wie else erweitert sie eine if-Anweisung um die Ausführung anderer Anweisungen, sobald die normale if-Bedingung als FALSE angesehen wird. Anders als bei else wird die Ausführung dieser anderen Anweisungen nur durchgeführt, wenn die bei elseif angegebene alternative Bedingung als TRUE angesehen wird. Der folgende Code wird z.B. a ist grösser als b, a ist gleich b oder a ist kleiner als b ausgeben:

```
if ($a > $b) {
  print "a ist grösser als b";
} elseif ($a == $b) {
  print "a ist gleich b";
} else {
  print "a ist kleiner als b";
}
```

Es kann mehrere elseif-Anweisungen innerhalb einer if-Anweisung geben. Die erste elseif-Bedingung (falls vorhanden), die true ist, wird ausgeführt. In PHP kann man auch 'else if' schreiben (zwei Wörter). Das Verhalten ist identisch zu 'elseif' (ein Wort), genau wie in C.
Die elseif-Anweisung wird nur ausgeführt, wenn die vorausgehende if-Bedingung sowie jede vorherige elseif-Bedingung FALSE ist und die aktuelle elseif-Bedingung TRUE ist.

9.4 Alternative Syntax für Kontroll-Strukturen

PHP bietet eine alternative Syntax für einige seiner Kontroll- Strukturen; als da sind if, while, for und switch. Die Grundform der alternativen Syntax besteht immer aus dem Wechsel der öffnenden Klammer gegen einen Doppelpinkt (:) und der schliessenden Klammer gegen ein endif;, endwhile;, endfor; bzw. endswitch;.

```
<?php if ($a == 5): ?>
  A ist gleich 5
<?php endif; ?>
```

Im obigen Beispiel ist der HTML-Bereich eingebettet in eine if-Anweisung mit alternativer Syntax. Der HTML-Bereich wird nur ausgegeben, wenn $a gleich 5 ist.
Die alternative Syntax kann auch auf else und elseif angewendet werden. Es folgt eine if-Struktur mit elseif und else im alternativen Format:

```
if ($a == 5):
  print "a ist gleich 5";
  print "...";
elseif ($a == 6):
  print "a ist gleich 6";
  print "!!!";
else:
  print "a ist weder 5 noch 6";
endif;
```

9.5 while

While-Schleifen sind die einfachste Form von Schleifen in PHP. Sie funktionieren genau wie in C. Die Grundform einer while-Anweisung lautet:

```
while (expr) Anweisung
```

Die Bedeutung einer while-Anweisung ist einfach. Sie weist PHP an, einen in ihr eingebetteten Befehl so lange zu wiederholen, bis die while- Bedingung TRUE geworden ist. Der Wert der Bedingung wird immer am Anfang der Schleife geprüft. Wird der Wert während der Ausführung der Befehle innerhalb der while-Schleife geändert, endet die Ausführung diese Befehls-Blocks nicht vor einem neuem Schleifen-Durchlauf (Iteration). Jeder Schleifendurchlauf ist eine Iteration. Falls die while- Bedingung bereits zu Beginn FALSE ist, werden die Anweisungen der while-Schleife nicht ein einziges Mal durchlaufen.

Wie bei der if-Anweisung kann man mehrere Befehle innerhalb einer Schleife angeben, indem man sie mit geschweiften Klammern umschliesst oder die alternative Syntax gebraucht:
while (expr): statement ... endwhile;

Die folgenden Beispiele sind identisch; beide geben Zahlen von 1 bis 10 aus:

```
/* Beispiel 1 */

$i = 1;
while ($i <= 10) {
  print $i++;  /* es wird erst $i ausgegeben,
                  bevor der Wert erhöht wird
                  (post-increment) */
}

/* Beispiel 2 */

$i = 1;
while ($i <= 10):
  print $i;
  $i++;
endwhile;
```

9.6 do..while

Do..while-Schleifen sind den while-Schleifen sehr ähnlich, ausser dass die Erfüllung des Ausdrucks am Ende jedes Durchlaufs geprüft wird (statt am Anfang). Der Hauptunterschied zu gewöhnlichen while-Schleifen ist, dass der erste Schleifen- Durchlauf bei do..while in jedem Fall statt findet, wogegen es bei while-Schleifen durchaus passieren kann, dass die Schleife nie durchlaufen wird, wenn die am Anfang zu prüfende Bedingung schon zu Beginn FALSE ist.
Es gibt nur eine Syntax für do..while-Schleifen:

```
$i = 0;
do {
  print $i;
} while ($i>0);
```

Die obige Schleife wird genau einmal durchlaufen, da nach der ersten Wiederholung die Erfüllung der Bedingung geprüft wird. Da diese aber nicht erfülltist, also FALSE ist ($i ist nicht grösser als 0), wird die Schleifenausführung beendet.
Erfahrene C-Anwender kennen auch die Möglichkeit, Programm-Blöcke mit do..while(0) einzuschliessen und dann die break Anweisung zu benutzen. Der folgende Programm-Ausschnitt zeigt dies:

```
do {
  if ($i < 5) {
    print "i ist nicht gross genug";
    break;
  }
  $i *= $factor;
  if ($i < $minimum_limit) {
    break;
  }
  print "i ist ok";
```

```
    ...process i...

} while(0);
```

Es ist nicht weiter tragisch, wenn sie dieses Beispiel nicht oder nur zum Teil verstehen. Sie können auch ohne dieses Feature effektive PHP-Programme und Skripte schreiben.

9.7 for

For-Schleifen sind die komplexesten Schleifen in PHP. Sie funktionieren wie ihr Gegenstück in C. Die Syntax einer for-Schleife sieht so aus:

```
for (expr1; expr2; expr3) Anweisung
```

Der erste Parameter (*expr1*) wird beim Schleifenbeginn geprüft bzw. ausgeführt (ohne jegliche Vorbedingung). Zu Beginn jedes Durchlaufs wird nun *expr2* geprüft. Wenn dieser TRUE ist, fährt die Schleife weiter fort mit der Ausführung der nachfolgenden Befehle. Wenn das Ergebnis FALSE lautet, wird die Schleife beendet.

Am Ende jedes Durchlaufs wird nun auch noch *expr3* geprüft / ausgeführt.
Jeder der Parameter kann leer sein (optional). Ist *expr2* leer, wird die Schleife unbestimmt oft durchlaufen, da PHP ihn als TRUE wertet (wie in C). Das ist nicht so sinnlos, wie sie vielleicht glauben, weil man häufig eine Schleife erst durch eine bedingte break- Anweisung statt durch eine unwahr werdende for- Bedingung beenden möchte. Beachten sie die folgenden Beispiele. Alle geben Zahlen von 1 bis 10 aus:

```
/* Beispiel 1 */

for ($i = 1; $i <= 10; $i++) {
  print $i;     }
 /* Beispiel 2 */

for ($i = 1;;$i++) {
  if ($i > 10) {
    break;
  }
  print $i;
}


/* Beispiel 3 */

$i = 1;
for (;;) {
  if ($i > 10) {
    break;
  }
  print $i;
  $i++;
}

/* Beispiel 4 */

for ($i = 1; $i <= 10; print $i, $i++) ;
```

Selbstverständlich sieht das erste (oder das vierte) Beispiel am besten aus, aber sie werden noch feststellen, dass es oftmals ganz nützlich sein kann, leere Parameter in for- Schleifen zu verwenden.
PHP unterstützt auch bei for-Schleifen die alternative "Doppelpunkt-Syntax".

```
for (expr1; expr2; expr3): Anweisung; ...; endfor;
```

Andere Sprachen haben für das Durchlaufen eines Hash´s oder Arrays eine foreach-Anweisung. PHP3 hat dies nicht; im Gegensatz zu PHP4 (vgl.foreach). In PHP3 kann man dafür eine Kombination von while mit der list- und each-Funktion einsetzen. Beispiele finden sie in der Dokumentation zu diesen Funktionen.

9.8 foreach

PHP4 (nicht PHP3) enthält eine foreach-Funktion, genau wie Perl und andere Sprachen. Diese ermöglicht es, auf einfache Weise ein Array zu durchlaufen. Es gibt zwei Syntax- Formen; die zweite ist eine unbedeutende aber sinnvolle Erweiterung der ersten Syntax:

```
foreach(array_expression as $value) Anweisung
foreach(array_expression as $key => $value) Anweisung
```

Die erste Form durchläuft das array_expression- Array. Bei jedem Durchgang wird der Wert des aktuellen Elements $value zugewiesen und der interne Array-Zeiger um 1 erhöht. Dadurch wird beim nächsten Durchgang automatisch das nächste Element ausgewertet.
Die zweite Form arbeitet genauso, ausser dass bei jedem Durchlauf auch der aktuelle Schlüssel der Variablen $key zugewiesen wird.
Sobald foreach den ersten Durchlauf startet, wird er interne Array-Zeiger auf das erste Array-Element gesetzt. Deshalb brauchen sie nicht den Befehl reset vor Aufruf der foreach-Funktion abzusetzen.
Sie sollten beachten, dass die folgenden Beispiele in ihrer Funktionalität identisch sind:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
  echo "Wert: $value<br>\n";
}

foreach ($arr as $value) {
  echo "Wert: $value<br>\n";
}
```

Auch hier funktioniert alles gleich:

```
reset ($arr);
while (list($key, $value) = each ($arr)) {
  echo "Schlüüsel: $key; Wert: $value<br>\n";
}

foreach ($arr as $key => $value) {
  echo "Schlüssel: $key; Wert: $value<br>\n";
}
```

Noch einige Beispiel, die die Anwendung verdeutlichen:

```
/* foreach Beispiel 1: Nur der Wert */

$a = array (1, 2, 3, 17);

foreach ($a as $v) {
  print "Aktueller Wert von \$a: $v.\n";
}

/* foreach Beispiel 2: Wert (mit Ausgabe des Array-Schlüssels) */

$a = array (1, 2, 3, 17);

$i = 0; /* nur für Anschauungs-Zweck */

foreach($a as $v) {
  print "\$a[$i] => $k.\n";
}

/* foreach Beispiel 3: Schlüssel und Wert */

$a = array (
      "one" => 1,
      "two" => 2,
      "three" => 3,
```

```
    "seventeen" => 17
);

foreach($a as $k => $v) {
  print "\$a[$k] => $v.\n";
}
```

9.9 break

Break bricht die Ausführung der aktuellen if, for, while oder switch Anweisungs-Sequenz ab.
Einem break kann optional ein numerisches Argument angehängt werden, das die Anzahl der abzubrechenden Befehls-Sequenzen enthält.

```
$i = 0;
while ($i < 10) {
  if ($arr[$i] == "stop") {
    break;  /* Man kann hier auch 'break 1;' schreiben. */
  }
  $i++;
}

/* Benutzung des optionalen Argumentes. */

$i = 0;
while ( ++$i ) {
  switch ( $i ) {
    case 5:
      echo "Bei 5<br>\n";
      break 1;  /* Beendet nur switch. */
    case 10:
      echo "Bei 10; quitting<br>\n";
      break 2;  /* Beendet switch und while. */
    default:
      break;
  }
}
```

9.10 continue

Continue wird innerhalb von Schleifen gebraucht. Die Schleife wird an der aktuellen Stelle abgebrochen und es wird der nächste Durchlauf begonnen.
Continue kann optional ein numerisches Argument erhalten, das die Anzahl der zu überspringenden Schleifendurchläufe beinhaltet.

```
while (list ($key, $value) = each ($arr)) {
  if ($key % 2) { // überspringe gerade Werte
    continue;
  }
  tue_was_mit_ungerade ($value);
}

$i = 0;
while ($i++ < 5) {
  echo "Aussen<br>\n";
  while (1) {
    echo "  Mitte<br>\n";
    while (1) {
      echo "  Innen<br>\n";
      continue 3;
    }
```

```
    echo "Das wird nie ausgegeben.<br>\n";
  }
  echo "Dies auch nicht.<br>\n";
}
```

9.11 switch

Die switch-Anweisung ist gleichbedeutend einer Reihe von IF-Anweisungen mit dem gleichen Parameter. Häufig wollen sie ein und die selbe Variable (bzw. denselben Ausdruck) mit verschiedensten Werten vergleichen und in Abhängigkeit vom Auswertungsergebnis verschiedene Programmteile ausführen. Genau das ermöglicht die switch-Anweisung.
Die folgenden 2 Beispiele zeigen 2 verschiedene Wege, das gleiche zu bewirken; eins gebraucht mehrere if-Befehle, das andere eine switch-Anweisung:

```
if ($i == 0) {
  print "i ist gleich 0";
}
if ($i == 1) {
  print "i ist gleich 1";
}
if ($i == 2) {
  print "i ist gleich 2";
}

switch ($i) {
  case 0:
     print "i ist gleich 0";
     break;
  case 1:
     print "i ist gleich 1";
     break;
  case 2:
     print "i ist gleich 2";
     break;
}
```

Es ist wichtig, die Ausführung einer switch- Anweisung zu verstehen, um Fehler zu vermeiden. Die switch-Anweisung wird Zeile für Zeile (also Anweisung für Anweisung) abgearbeitet. Zu Beginn wird nichts ausgeführt. Erst wenn bei einem case-Teil eine Entsprechung zum switch-Ausdruck vorliegt, werden die darin enthaltenen Befehle ausgeführt. PHP fährt dann mit der Abarbeitung des restlichen Codes innerhalb des switch-Blocks fort (oder bis zum ersten Auftreten einer break-Anweisung). Ohne break am Ende eines case-Teils werden also noch die folgenden case-Blöcke ausgeführt. Z.B.:

```
switch ($i) {
  case 0:
     print "i ist gleich 0";
  case 1:
     print "i ist gleich 1";
  case 2:
     print "i ist gleich 2";
}
```

Wenn hier $i gleich 0 ist, würde PHP alle print-Anweisungen ausführen. Ist $i gleich 1, würden die letzten beiden print- Befehle ausgeführt und wenn $i = 2 ist, würde nur der letzte print-Befehl ausgeführt. Deshalb ist es wichtig, bei der ersten gefundenen Übereinstimmung eine break-Anweisung zu setzen (abgesehen von bestimmten Fällen, wo genau dieses Verhalten gefordert wird).
Bei einer switch-Anweisung wird die Bedingung also nur einmal überprüft und das Ergebnis mit jeder case-Anweisung verglichen. Bei einem elseif-Befehl wird die Bedingung neu geprüft. Eine switch-Anweisung kann schneller als mehrere if-Befehle sein, z.B. bei komplizierteren Bedingungen als einem einfachen Vergleich.
Der Anweisungsteil von case kann auch leer sein. Dann wird die Kontrolle einfach an den nächsten case-Teil übergeben.

```
switch ($i) {
  case 0:
  case 1:
  case 2:
     print "i ist kleiner als 3 aber nicht negativ";
     break;

  case 3:
     print "i ist gleich 3";
}
```

Ein Spezialfall ist die Anweisung default. Diese trifft auf alles zu, was nicht von den voranstehenden case-Ausdrücken erfasst wurde, wie z.B.:

```
switch ($i) {
  case 0:
     print "i ist gleich 0";
     break;
  case 1:
     print "i ist gleich 1";
     break;
  case 2:
     print "i ist gleich 2";
     break;
  default:
     print "i ist weder 0, 1 noch 2";
}
```

Der case-Ausdruck kann eine Prüfung einfacher Typen sein, also von Integer- oder Fließkomma-Zahlen oder von Strings / Zeichenketten. Arrays oder Objekte können nicht benutzt werden, es sei denn sie werden auf einfache Typen herunter gebrochen.

```
switch ($i):
  case 0:
     print "i ist gleich 0";
     break;
  case 1:
     print "i ist gleich 1";
     break;
  case 2:
     print "i ist gleich 2";
     break;
  default:
     print "i iste weder 0, 1 noch 2";
endswitch;
```

9.12 require

Der require-Befehl setzt an seine Stelle den Inhalt der angegebenen Datei (ähnlich dem #include von C). Wichtig: PHP kehrt zu Beginn der per include oder require eingebundenen Dateien vom PHP- in den HTML-Modus und am Schluß der Datei wieder vom HTML- in den PHP-Modus zurück. Falls innerhalb dieser Dateien also PHP- Code ausgeführt werden soll, muss dieser eingeschlossen werden von gültigen PHP-Start- und PHP-Ende-Marken. Require ist keine PHP-Anweisung, sondern eine spezielle Sprachanweisung. Sie ist anderen Regeln unterworfen als Funktionen. Einerseits unterliegt sie keinen Kontroll-Srukturen, andererseits gibt sie keinen Wert zurück. Der Versuch, von einem require-Aufruf einen Rückgabewert zu erhalten, führt zu einem Parse-Fehler.
Anders als include wird require immer die angegebene Datei einlesen, auch dann, wenn die Programmzeile, in der sie steht, nicht ausgeführt wird. Wenn sie eine Datei nur bedingt einlesen wollen, benutzen sie include. Die bedingte Anweisung würde require nicht davon abhalten, die Datei zu laden.
Innerhalb von Schleifen tritt der Effekt auf, dass, obwohl die den require-Befehl enthaltende Zeile mehrfach angesprungen wird, die entsprechende Datei trotzdem nur genau einmal eingelesen (ausgeführt) wird.
Sie können also require nicht innerhalb von Programm-Schleifen einsetzen. Deshalb, und wenn sie verschiedene Dateien einlesen wollen, müssen sie in Schleifen den Befehl include benutzen.

```
require ('header.inc');
```

Beachten sie, dass sowohl include als auch require den Inhalt der angegebenen Datei in das Skript nur 1:1 einlesen. Sie kann weder über HTTP noch sonstwie eingelesen werden. Jede vor dem Einbinden der Datei definierte Variable ist innerhalb dieser Datei verfügbar, da sie Teil des umgebenden Skripts wird.

```
require ("file.inc?vareins=1&varzwei=2"); /* Funkt. nicht. */

$vareins = 1;
$varzwei = 2;
/* $vareins und $varzwei sind in file.inc verfügbar */
require ("file.inc");
```

9.13 include

Die include-Anweisung liest die angegebene Datei ein und wertet sie aus.
Wichtig ist, dass beim include- oder require-Befehl vom PHP-Parsing-Modus in den HTML-Modus geschaltet wird und bei Rückkehr in das aufrufende Skript wieder zurück vom HTML- in den PHP-Modus. Deshalb muss jeder PHP- Code innerhalb der eingebundenen Dateien umschlossen werden.

Das passiert jedes Mal, sobald die include- Anweisung auftritt. Deshalb können sie include auch sehr gut innerhalb von Schleifen verwenden, um eine Anzahl unterschiedlicher Dateien einzubinden.

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) {
  include $files[$i];
}
```

Include unterscheidet sich von require dadurch, dass die include-Anweisung jedesmal neu ausgewertet wird, sobald sie auftritt (und nur zur Ausführungszeit). Dagegen wird die require- Anweisung beim ersten Auftreten mit der angegebenen Datei ersetzt, egal ob sie einzubinden ist oder nicht (innerhalb von bedingten Anweisungen, z.B. bei if auch dann, wenn das ausgewertete Argument false ergeben hat).
Da include ein spezieller Sprach-Konstrukt ist, müssen sie ihn innerhalb einer bedingten Anweisung in einen Anweisungsblock setzen.

```
/* Das ist falsch und führt nicht zum gewünschten Ergebnis. */

if ($Bedingung)
  include($diesedatei);
else
  include($anderedatei);

/* Diese ist korrekt. */

if ($Bedingung) {
  include($diesedatei);
} else {
  include($anderedatei);
}
```

Sowohl in PHP3 als auch in PHP4 ist es möglich, eine return-Anweisung innerhalb einer includeeten Datei anzugeben, um die Ausführung dieser Datei abzubrechen und zum aufrufenden-Skript zurück zu kehren. Es gibt aber ein paar Unterschiede in der Arbeitsweise. Der erste ist, dass bei PHP3 die return-Anweisung nicht innerhalb eines Blocks auftreten darf, es sei denn, es ist ein Funktions-Block. In diesem Fall gilt return für diese Funktion und nicht für die ganze Datei. In PHP4 gibt es diese Beschränkung nicht. PHP4 erlaubt ihnen auch die Rückgabe von Werten bei includeeten Dateien. Sie können den Wert des include-Aufrufs nutzen, als wenn sie eine Funktion aufgerufen hätten. Das wird in PHP3 eine Parse-Error ergeben.

Beispiel: Include in PHP3 und PHP4
Beachten sie, dass die folgende Datei, genannt test.inc) im gleichen Verzeichnis wie die Hauptdatei stehen muss:

```
<?php
echo "Vor dem Return <br>\n";
if (1) {
    return 27;
```

```
}
echo "Nach dem Return <br>\n";
?>
```

Die Datei main.html enthält folgendes:

```
<?php
$retval = include ('test.inc');
echo "Datei gibt zurück: '$retval'<br>\n";
?>
```

Sobald main.html in PHP3 aufgerufen wird, wird ein Parse-Error in Zeile 2 angezeigt; sie können in PHP3 also keinen Rückgabewert von include erhalten. In PHP4 wird das Ergebnis sein:

```
Vor dem Return
Datei gibt zurück: '27'
```

Die Datei main.html soll nun folgendes enthalten:

```
<?php
include ('test.inc');
echo "Zurück in main.html<br>\n";
?>
```

In PHP4 wird die Ausgabe sein:

```
Vor dem Return
Zurück in main.html
```

PHP3 wird dagegen folgendes ausgeben:

```
Vor dem Return
27Zurück in main.html

Parse error: parse error in /home/torben/public_html/phptest/main.html on line 5
```

Der obige Parse-Error ist das Ergebnis der Tatsache, dass die return-Anweisung innerhalb eines Nicht-Funktions-Blocks von test.inc steht. Wenn das return ausserhalb diese Blocks zu stehen kommt, wird die Auagabe wie folgt aussehen:

```
Before the return
27Back in main.html
```

Die '27' entsteht aus der Tatsache, dass PHP3 keine solchen Rückgabewerte unterstützt.
Beachten sie, dass sowohl include als auch require den Inhalt der einzufügenden Datei an ihre eigene Stelle des Skripts setzen. Sie lesen diese Datei nicht über HTTP oder sonstiges ein. Deshalb ist jede in der Hauptdatei gesetzte Variable auch innerhalb der eingebundenen Datei vorhanden; sie ist ja Bestandteil dieser Datei geworden.

```
include ("file.inc?vareins=1&varzwei=2"); /* Funkt. nicht. */

$vareins = 1;
$varzwei = 2;
/* $vareins und $varzwei sind in file.inc verfügbar */
include ("file.inc");
```

Lassen sie sich nicht durch die Tatsache irritieren, dass sie Dateien auch über HTTP per require oder include einbinden können (vgl. Remote files-Möglichkeit). Das oben gesagte behält seine Gültigkeit.

10. Funktionen

10.1 Vom Nutzer definierte Funktionen

Eine Funktion kann wie folgt definiert werden:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
  echo "Beispielfunktion.\n";
  return $retval;
```

```
}
```

Jeder beliebige korrekte PHP-Code kann in einer Funktion vorkommen, sogar andere Funktionen und Klassen-Definitionen In PHP3 müssen Funktionen definiert sein, bevor man auf sie verweist. In PHP4 ist das nicht mehr erforderlich.

10.2 Funktionsparameter

Mit einer Parameterliste kann man Informationen an eine Funktion übergeben. Die Parameterliste ist eine kommagetrennte Liste von Variablen und/oder Konstanten.
PHP unterstützt die Weitergabe von Parametern als Werte (das ist der Standard), als Verweise , und als Vorgabewerte . Die Übergabe einer beliebigen Parameterzahl ist nicht möglich, einen ähnlichen Effekt kann man aber durch die Übergabe von Arrays erreichen.

```
function rechne_array($eingabe) {
   echo "$eingabe[0] + $eingabe[1] = ", $eingabe[0]+$eingabe[1];
}
```

10.3 Verweise als Parameter übergeben

Normalerweise werden den Funktionen Werte als Parameter übermittelt. Wenn man den Wert dieses Parameters innerhalb der Funktion ändert, bleibt der Parameter ausserhalb der Funktion unverändert. Wollen Sie aber genau das erreichen, dann müssen Sie die Parameter als Verweise übergeben.
Wenn eine Funktion einen Parameter generell als Verweis behandeln soll, setzt man ein kaufmännisches Und(&) in der Funktionsdefinition vor den Parameternamen:

```
function fuege_etwas_anderes_an(&$string) {
   $string .= 'und nun zu etwas völlig anderem.';
}
$str = 'Dies ist ein String, ';
add_some_extra($str);
echo $str; // Ausgabe 'Dies ist ein String, und nun zu etwas völlig anderem.'
```

Wenn Sie eine Variable andererseits als Verweis an eine Funktion übergeben wollen, die dies nicht tut, können Sie das kaufmännische Und(&) auch beim Aufruf der Funktion verwenden:

```
function bla ($fasel) {
   $fasel .= ' und nun zu etwas völlig anderem.';
}
$str = 'Dies ist ein String, , ';
bla ($str);
echo $str;  // outputs 'Dies ist ein String, '
bla (&$str);
echo $str;  // outputs 'Dies ist ein String, und nun zu etwas völlig anderem.'
```

10.4 Vorgabewerte für Parameter

Eine Funktion kann c++-artige Vorgabewerte für skalare Parameter wie folgt definieren:

```
function machkaffee ($typ = "Cappucino") {
   return "Ich mache eine Tasse $typ.\n";
}
echo machkaffee ();
echo makecoffee ("Espresso");
```

Die Ausgabe von diesem kleinen Skript ist:
Ich mache eine Tasse Cappucino.
Ich mache eine Tasse Espresso.

Der Vorgabewert muß eine Konstante sein, darf also (zum Beispiel) keine Variable oder Element einer Klasse sein.

In PHP 4.0 ist es ebenfalls möglich, unset als Vorgabewert zu definieren. Dann wird der Parameter auf gar keinen Fall gesetzt, wenn er bisher keinen Wert hat.
Bitte beachten Sie, dass alle Vorgabewerte rechts von den Nicht-Vorgabeparametern stehen sollten; - sonst wird es nicht funktionieren. Betrachten Sie folgendes Beispiel:

```
function mach_joghurt ($typ = "rechtsdrehendes", $geschmack) {
   return "Mache einen Becher $typ $geschmack-joghurt.\n";
}

echo mach_joghurt ("Brombeer");   // arbeitet nicht wie erwartet
```

Die Ausgabe dieses Beispiels ist::

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Mache einen Becher Brombeer-joghurt.
```

Nun vergleichen Sie bitte oberes Beispiel mit folgendem:

```
function mach_joghurt ($geschmack, $typ = "rechtsdrehendes") {
   return "Mache einen Becher $typ $geschmack-joghurt.\n";
}

echo mach_jogurt ("Brombeer");   // arbeitet wie erwartet.
```

... und jetzt ist die Ausgabe:
Mache einen Becher rechtsdrehendes Brombeer-Joghurt.

10.5 Rückgabewerte

Sie können Werte mit dem optionalen Befehl "return" zurückgeben. Es können Variablen jedes Typs zurückgegeben werden, auch Listen oder Objekte.

```
function quadrat ($zahl) {
   return $zahl * $zahl;
}
echo quadrat (4);   // gibt '16' aus.
```

Es ist leider nicht möglich, mehrere Werte von einer Funktion zurückgeben zu lassen. Ein ähnliches Resultat kann man aber durch die Rückgabe von Listen bekommen.

```
function kleine_zahlen() {
  return array (0, 1, 2);
}
list ($null, $eins, $zwei) = small_numbers();
```

10.6 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 Variable ist. Diese Möglichkeit kann für Callbacks, Funktionstabellen und

Beispiel für Variablenfunktionen

```
<?php
function blah() {
   echo "In blah()<br>\n";
}

function fasel( $arg = '' ) {
   echo "In fasel(); ist der Parameter '$arg'.<br>\n";
}
```

```
$func = 'blah';
$func();
$func = 'fasel';
$func( 'test' );
?>
```

11 Klassen und Objekte

11.1 Klassen

Eine Klasse ist eine Sammlung von Variablen und von Funktionen, die mit diesen Variablen arbeiten. Eine Klasse wird folgendermaßen definiert:

```
<?php
class Einkaufswagen {
   var $produkte;  // Produkte in Ihrem Einkaufswagen

   // Füge dem Einkaufswagen $anzahl Artikel der Sorte $artnr zu

   function fuege_hinzu ($artnr, $anzahl) {
      $this->produkte[$artnr] += $anzahl;
   }

   // Nimm $anzahl von Artikel wieder aus dem Einkaufswagen

   function nimm_heraus ($artnr, $anzahl) {
      if ($this->produkte[$artnr] > $anzahl) {
         $this->produkte[$artnr] -= $anzahl;
         return true;
      } else {
         return false;
      }
   }
}
?>
```

In diesem Beispiel wird eine Klasse "Einkaufswagen" definiert. Sie besteht aus einem assoziativen Array von Produkten im Einkaufswagen und zwei Funktionen zum Zufügen und Entfernen von Einkäufen.
Klassen sind Typen, dass heisst die Blaupausen für realle Variablen. Um sie zu nutzen, muß zunächst eine Variable mit dem Operator new angelegt werden.

```
$einkaufswagen = new Einkaufswagen;
$einkaufswagen->fuege_hinzu("10", 1);
```

Hier wird das Objekt $einkaufswagen aus der Klasse Einkaufswagen geschaffen. Dann wird die enthaltene Funktion fuege_hinzu() aufgerufen, um ein Produkt mit der Artikelnummer 10 in den Einkaufswagen zu tun.
Klassen können ebenfalls Erweiterungen von anderen Klassen sein. Die erweiterte, oder auch abgeleitete Klasse enthält alle Funktionen der ursprünglichen Klasse, und dazu die eigenen Ergänzungen. Das geschieht mit dem Schlüsselwort "extends". Mehrfachvererbung wird von PHP nicht unterstützt.

```
class Mein_Einkaufswagen extends Einkaufswagen {
   var $besitzer;

   function setze_besitzer ($name) {
      $this->besitzer = $name;
   }
}
```

In diesem Beispiel wird eine Klasse Mein_Einkaufswagen definiert. Sie enthält alle Funktionen der Klasse Einkaufswagen, eine zusätzliche Variable $besitzer und die zusätzliche Funktion setze_besitzer(). Man kann den Einkaufswagen auch weiterhin wie oben erzeugen, nur kann man jetzt auch den Besitzer setzen oder herausfinden. Die alten Funktionen der Klasse Einkaufswagen können ebenfalls weiterverwendet werden.

```
$meinkaufswagen = new Mein_Einkaufswagen;  // Kreiere einen Einkaufswagen
$meinkaufswagen->setze_besitzer ("kris");          // Name dieser Klasse
print $meinkaufswagen->besitzer;          // schreibe den Namen des Besitzers
$meinkaufswagen->fuege_hinzu("10", 1);  // (Siehe oben, vererbt von Einkaufswagen)
```

Innerhalb der Funktionen einer Klasse bezeichnet die Variable $this das aktuelle Objekt. Sie können mit $this->irgendwas auf dessen Variablen und Funktionen zugreifen.
Konstruktoren sind Funktionen eine Klasse, die beim Erschaffen eines neuen Objektes automatisch aufgerufen werden. Eine Funktion wird zu einem Kontruktor, wenn Sie den gleichen Namen wie die Klasse trägt.

```
class Auto_Einkaufswagen extends Einkaufswagen {
   function Auto_Einkaufswagen () {
      $this->fuege_hinzu ("10", 1);
   }
}
```

Die Klasse Auto_Einkaufswagen entspricht der Klasse Einkaufswagen plus einen Kontstruktor, der bereits für eine erste Füllung (1 Artikel der Nummer 10) gesorgt hat. Jeder neu erzeugte Auto_Einkaufswagen enthält so von vorneherein diesen Artikel. Kontruktoren können auch Parameter enthalten. Aber diese Parameter sind optional, und können so nützlicher eingesetzt werden.

```
class Konstruktor_Einkaufswagen extends Einkaufswagen {
   function Konstruktor_Einkaufswagen ($produkt = "10", $anzahl = 1) {
      $this->fuege_hinzu ($produkt, $anzahl);
   }
}

// Kaufe wieder den gleichen alten Kram ein.

$standard_einkaufswagen   = new Konstruktor_Einkaufswagen;

// Kaufe etwas bestimmtes ein ...

$anderer_Einkaufswagen = new Konstruktor_Einkaufswagen ("20", 17);
```

Achtung:
Bei abgeleiteten Klassen wird der Konstruktor der Ursprungsklasse nicht automatisch aufgerufen, wenn der Konstruktor der abgeleiteten Klasse aufgerufen wird.

Übung: Link-Liste

In einer Datei sollen Links gespeichert werden, die dann auf einer Webseite mittels eines PHP-Scriptes angezeigt werden. Die Datei enthält also Zeilen mit Adressen wie folgt:

http://www.web.de
http://www.lycos.de

Jeder Benutzer soll neue Links in die Liste eintragen können. Dazu sind folgende Komponenten nötig:

- PHP-Script, das die Links aus der Datei liest und im Browser anzeigt.
  Weiterhin soll entweder am Anfang oder am Ende der Liste ein Formular mit einem Eingabefeld und einem Submit-Button angezeigt werden. Hier kann der Benutzer einen neuen Link eingeben. Beim Drücken des Buttons wird das nachfolgende Script aufgerufen und der eingegebene Link in der Datei gespeichert.

- PHP-Script, das einen neuen Link in die Datei hineinschreibt. Der Link wird wieder an das Ende der Datei angehängt. (Modus "a" – Anhängen).

Anmerkung:
Verwenden Sie für die Übungen folgende PHP-Funktionen (siehe auch Doku Funktionen des Dateisystem und String-Funktionen):

fopen(Dateiname,Modus)	Datei öffnen, "r" (Lesen), "w" (Schreiben), "a" (Anhängen),
fclose(Handle)	Datei schliessen
feof(Handle)	Prüfung auf Dateiende
fgets(Handle, maxzeichen);	Zeile aus Datei lesen
fputs(Handle,String);	String in Datei schreiben