Doing Web Apps

PHP Sprachkurs

Autor: Rüdiger Marwein
Letzte Änderung: 2012-10-18
Version: 0.9

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument setzt nach der Installation von PHP und MySQL an.

Es führt in die Sprachmerkmale von PHP ein.

Es soll sowohl die grundlegende Syntax als auch die häufig verwendeten Funktionen erklären. Auch häufige Fehler und Fehlermeldungen sollen angesprochen werden.

Praxisnahe Beispiele werden hier nicht zu finden sein, da es ja erstmal darum geht, die Syntax kennen zu lernen.

Lesen Sie das Dokument ein mal sorgfältig, lassen sich aber nicht von (noch) unbekannten Ausdrücken aufhalten.
Lesen sie es danach noch einmal, nun sollten die ehemals unklaren Stellen leicht verständlich sein.

2. Grundlegende Syntax

2.1. Allgemeines

PHP-Bereiche (auch wenn kein HTML geschrieben wird) werden immer in

```
<?php [...] ?>
```

gefasst. Diese kennzeichnen jeweils den Bereich, der von PHP ausgeführt werden soll. Was sich außerhalb der Bereiche befindet, wird einfach ausgegeben. So ein Bereich kann sich mitten in einer Zeile HTML befinden und sogar vom Anfang des Dokuments bis zum Ende gehen.

Jedes PHP Kommando endet mit einem ; (Strichpunkt), außer es handelt sich um { ... } (Bereichsklammern). Kommandos können über mehrere Zeilen verteilt werden. Zeichenketten ebenfalls.

```
<?php
if( $dieEineVariable &&
    $dieAndereVariable ) {
  echo "Hallo Welt";
  $dieAndereVariable = true;
}
?>
```

<?php und ?> werden nachfolgend aus Platzgründen in den Beispielen **nicht mehr mit aufgeführt**, außer es tritt eine Vermischung mit HTML auf.

Zeichenketten (Strings) können in einfachen Hochkomma oder Gänsefüßchen geschrieben werden. Der Unterschied ist, dass einfache Hochkomma **keine Inline-Variablen** und **kein Steuerzeichen** erkennen. Mehrere Strings können mit dem **Punkt-Operator** vereinigt werden.

Steuerzeichen beispiele aus der Praxis:

\n = newline / Zeilenumbruch
\r = carriage return / „Wagenvorlauf“

\t = tab / Tab

```
$var = "Variable";
echo "Dies ist ein String mit\n\t".
     "Steuerzeichen und \n\teiner ".
     "integrierten \n\t$var";
```

Ergibt die Ausgabe:

```
Dies ist ein String mit
    Steuerzeichen und
    einer integrierten
    Variable
```

Groß- und Kleinschreibung ist bei **Funktions- und Klassennamen** egal. Bei **Variablennamen** wird sie allerdings berücksichtigt.

Kommentare können 3 Formen haben:

```
/* Bereichs-Kommentar kann über
mehrere Zeilen gehen */

// Zeilen-Kommentar geht bis zum Ende der Zeile

# Zeilen-Kommentar geht bis zum Ende der Zeile
```

```
/**
 * PHP Doc Kommentar (ähnlich JavaDoc)
 *
 * Dies ist eine spezielle Art von Kommentaren, aus denen
 * Dokumentationen erstellt werden können
 *
 * @author Rüdiger Marwein
 */
```

2.2. Variablen

Bei PHP beginnen Variablen immer mit einem `$` (Dollar-Zeichen). Das hat zwei Gründe. Zum einen kann nach einer mit Dollar ausgezeichneten Variable schneller geparst werden (für den Interpreter) und zum zweiten kann man so leicht Variablen in Zeichenketten unterbringen.

```
$name = "Peter Müller";
$name = "Ihr Name: $nachname, $vorname";
```

Bei PHP wird **Typsicherheit klein geschrieben**. Eine Variable ist ab dem Moment vorhanden, in dem sie **das erste mal** benutzt wird. Der ihr zugewiesene Wert bestimmt den **vorübergehenden Typ**. Wird ihr ein neuer Wert zugewiesen wird dessen Typ verwendet.

Anhand des Inhaltes wird **versucht einen Typ festzulegen**. Je nachdem welche Aktion ausgeführt wird (String-Ausgabe / Berechnung / Boolsche Abfrage), kann eine Variable anders ausgelegt werden.

Bestes Beipiel ist eine Boolsche Abfrage

```
if($gesetzt) echo "Hallo Welt";
```

wenn `$gesetzt` einen dieser Werte hat:

`false / null / 0 / '0' / '' (Leerstring) / neue Variable`

Die Ausgabe erfolgt **nie**, da alle Werte als boolsches `false` ausgewertet werden.

2.2.1. Variablen-Scopes

Variablen haben in der Regel einen Herkunftsort. Sie können lokale oder globale Variablen sein und es können Funktionsparameter sein. Neben diesen „Herkunftsorten“ kennt PHP noch einige mehr. Es gibt Werte, die an die URL angehängt werden (**GET**) und in PHP abfragbar sind.

`http://localhost/index.php`**`?parameter=wert&test=hallo`**

Diese hier übergebenen Variablen sind innerhalb der PHP-Umgebung in dem **immer verfügbaren** Array

`$_GET`

zu finden. Man kann ihn so auslesen: `$_GET['parameter']`

Werden Werte über ein Formular mit der Methode **POST** übergeben, landen sie in dem Array

`$_POST`

und sind genauso mit `$_POST['parameter']` abfragbar.

Genauso funktioniert es mit Werten die aus **Cookies** und aus der laufenden Benutzer-**Session** ausgelesen werden.

Diese befinden sich in

`$_COOKIE`

und in

`$_SESSION`.

Zusätzlich gibt es noch das Array

`$GLOBALS`

(ohne Unterstrich vorne), mit dem direkt auf **globale** Variablen zugegriffen werden kann, ohne sie z.B. in einer Funktion mit dem Schlüsselwort `global` erst sichtbar zu machen.

Und für die Variablen, die Einstellungen und Umgebungsvariablen des Webservers beinhalten gibt es das Array

`$_SERVER`.

In diesem wird vor allem `$_SERVER['DOCUMENT_ROOT']` häufig benötigt, da es das htdocs-Verzeichnis mit komplettem Pfad beinhaltet. Um die Werte mal in Aktion zu sehen, einfach eine PHP-Datei mit `<?php phpinfo(); ?>` erstellen und aufrufen.

Die o.g. Arrays sind alle **immer global verfügbar** (Superglobals) und müssen nicht mit

`global` sichtbar gemacht werden. Sie sind einfach da.

Zu den Scopes GET und POST siehe **Formularverarbeitung mit PHP**.

Zu den Scopes COOKIE und SESSION siehe **Persistente Daten verwalten mit PHP**.

2.3. Arrays

Arrays sind des PHP-Programmierers liebstes Spielzeug. Mit ihnen lassen sich Werte bündeln, Bäume aufbauen, Listen simulieren und sie lassen sich sogar als Stack und Queue verwenden. Eine definierte maximale Größe kann nicht angegeben werden, sie wachsen dynamisch mit.

```
// einige mögliche Deklarationen
// leeres Array
$benutzer = Array();
/* Array der Form
      [0]='Peter'
      [1]='Müller'       */
$benutzer = Array('Peter','Müller');
/* Hash-Array der Form
      ['vorname'] = 'Peter'
      ['nachname'] = 'Müller' */
$benutzer = Array('vorname'=>'Peter','nachname'=>'Müller');
// Hinten anhängen, Key wird automatisch erhöht
$benutzer[] = 'Schmitt';

// Mehrere Dimensionen ansprechen
$mehrdimensional[3][1]['key'] = 'Hallo';
```

Numerisches Array und Hash lassen sich beliebig kombinieren und beliebig tief in einander schachteln und Typen sind wie bereits gesagt völlig egal.

Zu Arrays gibt es eine ganze Latte Funktionen, von denen später einige wichtige vorgestellt werden.

2.4. Arithmetische Operatoren

Auch hier kommt es auf den in der Variable erkannten Typ an.

```
$x = 3;
$y = 9;

// plus (+), minus (-), mal (*), geteilt (/)
echo $x + $y - 3 * '7' / $x;

// modulo
echo $y % $x;

// inkrementieren / decrementieren
$x--; --$x;
$y++; ++$y;

// arithmetische Abkürzungen
$x+=2;  $y*=3;
$y-=2;  $x/=3; $y%=2;
```

Postinkrement (`$x++`) erhöht die Variable erst nach der aktuellen Verwendung. Analog das Postdecrement.

`$x+=2` ist gleichbedeutend mit `$x=$x+2`.

```
$x = 0;
echo $x++;    // -> 0
echo $x;      // -> 1
echo ++$x;    // -> 2
```

2.5. Boolsche Operationen

true ist alles was als ungleich 0 intepretiert werden kann. `false / null / 0 / '0' / '' (Leer-String) / neue Variable` wird alles als false interpretiert. Was liefert boolsch auswertbare Werte in PHP?

- Alles. Prinzipiell.
- Vergleiche
- Vorhanden-sein von **nicht-null-Inhalt** ist true.
- boolsche Verknüpfungen

Sollen Werte verglichen werden die per Definition schon mal aus Versehen beide als false interpretiert werden könnten, so kann das automatische Casting (Typanpassung) unterdrückt werden.

```
$a = 2;
$b = true;
// Gleichheit   || mit Typgleichheit
$a==$b;            $a===$b;
// Ungleichheit || mit Typgleichheit
$a!=$b;            $a!==$b;
// Grösser- kleiner
$a < $b;  $a <= $b;
```

```
$a > $b;  $a >= $b;
// Verknüpfungen
// und          || oder
$a==$b &&  $b      $a==$b || $b
$a==$b and $b      $a==$b or $b
// xor
$a xor $b;
```

2.6. Entscheidungen

Die syntaktische Struktur von Entscheidungen gestaltet sich sehr einfach, so dass hier nur ein Fallbeispiel dargestellt ist.

```
if($wert==1) {
  echo "ist gleich eins";
} elseif($wert==2) {
  echo "ist gleich zwei";
} else {
  echo "ist irgendwas anderes";
}
```

Wobei `if` als einzige Anweisung Pflicht ist, beliebig viele `elseif`'s kommen können und `else` optional immer am Ende steht. Mehrere If-Konstruktionen können selbstverständlich verschachtelt werden.

```
switch($wert) {
  case 1:
    echo "ist gleich eins";
    break;
  case 2:
    echo "ist gleich zwei";
    break;
  default:
    echo "ist irgendwas anderes";
}
```

Wobei falls kein `break` am Ende eines `case`'s steht, durch die folgenden `case`'s „durchgefallen“ wird, bis ein `break` kommt oder der Switch inklusive `default` durchlaufen wurde.

2.7. Ausgabe

Es gibt viele Funktionen die eine direkte Ausgabe erzeugen. Die am häufigsten verwendete ist `echo` bzw. `print`. Eine andere ist `printf`, die eine direkte Formatierung implementiert. Auch und gerade hier gelten die Regeln für doppelte und einfache Hochkomma.

```
echo  1234;
echo  "Hallo";
print("Test".'123');
$euro = 123.456789;
printf("Preis: %.2f Euro", $euro); // -> Preis: 123.45 Euro
```

Das verwendete %.2f bedeutet „Fließkommazahl mit 2 Nachkommastellen“. Diese aus C stammende Notation wird in der Regel als bekannt vorausgesetzt. Darum wird dies am

Ende bei den Funktionen nochmal angesprochen.

Hinweis: Ein PHP-Fehler erzeugt ebenfalls eine direkte Ausgabe. Das kann ungewollte Folgen haben.

Exkurs: Ein Webbrowser wertet ja im Prinzip **nur Text** aus. Aus diesem Grund ist PHP auch hauptsächlich darauf ausgelegt Text zu erzeugen und über den Webserver an den Endabnehmer zu verschicken. Man kann **HTML direkt in einen Ausgabestring einbetten** und z.B. über Schleifen eine HTML-Tabelle erzeugen.
Idealerweise sollte man HTML nicht mir PHP ausgeben, weil das aus 20 Zeilen Code mal schnell 200 machen kann, was ein Script unübersichtlich und über die Zeit unpflegbar / unwartbar macht.

2.8. Zeichenketten verbinden (Konkatenation)

Mit dem Punktoperator können Werte miteinander verbunden (konkateniert) werden. Auch hier ist es egal, welchen Typ sie haben.

```
$uno   = "Meine ";
$due   = 7;
$tres  = " Zwerge";
$quat  = $uno . $due;  // = verbinden
$quat .= $tres;        // = hinten dran hängen
echo $quat . "!!";     // -> Meine 7 Zwerge!!
```

2.9. Schleifen

Schleifen sind Wiederholungen von Anweisungsfolgen, die irgendwann abbrechen. Der Abbruch (boolscher Ausdruck) kann im Kopf oder im Fuss der Schleifenanweisung stehen und entscheidet so, ob die Schleife mindestens einmal oder keinmal durchlaufen wird.

2.9.1. While- / Do-Schleifen

Man kann es aussprechen als **„solange gilt, mache“** (while) und **„mache, und dann solange gilt“** (do-while)

```
$i = 0;
// kopfgesteuert
while($i<2) {
  $i++;
  [...]
}
// fussgesteuert
do {
  $i--;
  [...]
} while($i>0);
```

2.9.2. For-Schleife

Die Syntax kann man sprechen als **„beginne mit a=0 und mache, solange a<10 ist, die Operation a++ und den Block“**.

```
for($a=0 ; $a<10 ; $a++) {
  echo "Durchlauf: $a <br />";
}
```

2.10. Foreach-Schleife

Um ein Array zu durchlaufen kann man die while-Schleife nutzen. Eine beliebtere Methode ist die foreach-Schleife, was soviel heißt wie **„laufe über jedes einzeln drüber und gibs mir als...“**.

```
$namen = Array('Peter','Max','Hanne','Lulu');

// Array mit while-Schleife
while(list($key,$value) = each($namen)) {
  print "$key:$value/";
}

echo "\n<br />";
// foreach ohne key, nur value
foreach($namen as $value) {
  print $value . " ";
}
echo "\n<br />";

// foreach mit key und value
foreach($namen as $key => $value) {
  print $key . " : " . $value . "\n<br />";
}
```

Ausgabe:

```
0:Peter/1:Max/2:Hanne/3:Lulu/
Peter Max Hanne Lulu
0 : Peter
1 : Max
2 : Hanne
3 : Lulu
```

Wenn man Arrays mit **mehr als 2 Dimensionen** durchläuft, ist zu beachten, dass immer nur eine Dimension (Key-Reihe) durchlaufen wird. Ist ein herausgezogenes Value ebenfalls ein Array muss es **separat durchlaufen** werden.

2.11. Abgekürzt kodieren

Bei Abfragen und Schleifen können die Bereichsbegrenzer { ... } weggelassen werden, wenn es sich nur um eine Anweisungs-Zeile handelt.

```
foreach($array as $k=>$v)
  echo "$k = $v \n";
```

Es wird bis zum Ende der ersten Folge-Anweisung ausgeführt. Danach ist die Abfrage / Schleife vorbei.

Man kann If-Else-Abfragen in ternärer (dreiteiliger) Notation schreiben:

```
$istGesetzt = $x>0 ? 'ja' : 'nein';
```

Das ist gleichbedeutend mit

```
if($x>0) $istGesetzt = 'ja';
else $istGesetzt = 'nein';
```

Also „$istFaul = $stimmt ? 'Positiv' : 'Negativ'“

2.12. Eigene Funktionen

Eine Funktion ist genauso Typ-los wie eine Variable. Der Rückgabewert kann, muss aber nicht, ausgewertet werden... wenn es ihn überhaupt gibt.

```
function entferneLeerzeichen($zeichenkette) {
  $neuerString = '';
  for($i=0; $i<=strlen($zeichenkette); $i++) {
    if($zeichenkette{$i} != ' ')
      $neuerString .= $zeichenkette{$i};
  }
  return $neuerString;
}
echo entferneLeerzeichen(' H a l l o  ');
// Ausgabe: Hallo
```

2.13. Eigene Klassen

Einfach gesagt: Eine Klasse ist wie ein Programm in sich. Eine Klasse hat Variablen und Funktionen. Von einer Klassen können viele Instanzen erstellt werden, die ihre eigene Variablen-Werte haben. So eine Instanz heißt auch „Objekt“.

Klassen mit PHP zu erstellen ist sehr einfach. Das folgende Beispiel zeigt eine Klasse mit einem Konstruktor und einer setzen-Funktion für eine Member[1]-Variable. Der Konstruktor wird bei der Erzeugung eines Objekt mit dem `new` Operator automatisch ausgeführt und kann das Objekt initialisieren.

```
class Example {
  public $value;

  function __construct() {
    $this->setValue(0);
  }

  function setValue($num) {
    $this->value = $num;
  }
  function getValue() {
    echo $this->value;
  }
}
```

```
// Klasse verwenden
$exampleObject = new Example();
```

1Variable ist Bestandteil der Klasse

```
$exampleObject->setValue(5);
echo $exampleObject->getValue();
```

Da der Punkt-Operator schon für die Zeichenketten-Verbindung verbraten wurde, benutzt man in PHP den Pfeil-Operator (->) um auf Variablen und Methoden in einer Klasse zuzugreifen.

Innerhalb der Methoden einer Klasse wird `$this` verwendet um auf „hauseigenen“ Variablen und Methoden zuzugreifen.

2.13.1. Vererbung

PHP bietet nur einfache Vererbung, man kann also nur von einer Klasse gleichzeitig ableiten; mit dem Schlüsselwort `extends`. Hierbei übernimmt die Klasse alle Variablen und Methoden vom „Vater“ und überschreibt Methoden, wenn der gleiche Name verwendet wird.

```
class betterExample extends Example {

  function setValue($num) {
    if($num<100) {
      $this->value = $num;
    }
  }

}
```

2.13.2. Zugriffssteuerung

Bei PHP5 wurden die Schlüsselworte `private`, `protected` und `public` eingeführt, die zum objektorientierten Standard gehören.

`public` bedeutet „ist sogar von außen zugänglich“
`private` bedeutet „ist nur innerhalb dieser einen Klasse zugänglich, nicht in Ableitungen, nicht von außen“
`protected` bedeutet „ist innerhalb dieser Klasse und deren Ableitungen zugänglich, nicht von außen“

Zudem kann man Variablen als `static` deklarieren, was bedeutet, dass diese Variable über alle Objekte einer Klasse hinweg den selben Wert hat.

Methoden, die ohne Objekt aufrufbar sein sollen, muss das Schlüsselwort `static` vorangestellt werden.

```
static function Methode() { ... }
```

In statischen Methoden besteht kein Zugriff auf `$this` und nicht-statische Variablen.

Man kann in Funktionen die statisch aufgerufen werden mittels `Klasse::Methode();` trotzdem kein Objekt vorhanden ist, auf die statischen Member-Variablen zugreifen, mittels `self::$memberVariable`.

```
// Auszüge, nur exemplarisch
class bla {
  private    $name = '';
  protected  $geburtsjahr;
```

```
    public function getName() { return $this->name }
}
```

Mit `private` und `protected` kann man die Verwendung von bestimmten Features für anderen Programmierern versteckt bzw. verboten werden.

Exkurs: Leitet man eine Klasse ab, so spricht man von **Generalisierung**. Wird eine Klasse aus Objekten anderer Klassen zusammengebaut, so spricht man von **Aggregation.** Sind die Teile der Aggregation existentiell voneinander Abhängig, spricht man von **Komposition**. Diese Begriffe spielen in der UML und bei Design-Patterns eine Rolle.

3. Umgang mit Dateien

Auch dieses Thema ist in PHP wenig spektakulär.

Um den gesamten Inhalt einer Datei in eine Variable zu speichern, benötigt man lediglich eine Zeile:

```
$inhalt = file_get_contents('C:\autoexec.bat');
```

Ebenso kann der Variableninhalt mit einer Zeile in eine Datei geschrieben werden:

```
file_put_contents('C:\autoexec.bat',$inhalt);
```

Sind weitere Tätigkeiten notwendig kann auch anders auf die Datei zugegriffen werden.

`fopen()` liefert einen Datei-Pointer zurück, mit dem auf eine Datei zugegriffen werden kann. Der Pointer wird bei jeder Operation mit übergeben. Dabei wird der Zugriffsmodus beim Öffnen schon festgelegt.

r = read / lesen
w = write / schreiben
a = append / anhängen
r+ = read and write / lesen und schreiben

Die Beispiele zeigen den prinzipiellen Umgang.

```
$inhalt = '';

// Datei zum lesen öffnen

$fp = fopen($_SERVER['DOCUMENT_ROOT'].
            '/dateien/datei.xml','r');

// Inhalt in Variable auslesen

while($c = fread($fp,1024) {
     $inhalt .= $c;
}
fclose($fp);
```

```
$xml = "

<person>

     <name>Marwein</name>

     <vorname>Rüdiger</vorname>
</person>

";

// Datei zum schreiben öffnen

$fp = fopen($_SERVER['DOCUMENT_ROOT'].
            '/dateien/datei.xml','w');

// Zeichenkette als Dateiinhalt speichern

fwrite($fp,trim($xml));
fclose($fp);
```

Außerdem gibt es Funktionen wie **readfile()**, die den Inhalt der Datei einfach ausgeben,

ohne einen Datei-Pointer zu brauchen. Die Funktion **file()** liefert die Datei als Array mit einem Schlüssel für jede Zeile.

```
$inhalt = join(file($_SERVER['DOCUMENT_ROOT'].
                    '/dateien/datei.xml'));
```

Für den Umgang von z.B. aus Excel exportierten csv[2]-Dateien kann man die Funktion **fgetcsv()** verwenden, die die Felder pro Zeile in einem Array zurück gibt. Neben der maximalen Zeilenlänge wird auch das Trennzeichen der Felder angegeben. In diesem Fall das Komma als dritter Parameter.

```
$inhalt = '';

// Datei zum lesen öffnen

$fp = fopen($_SERVER['DOCUMENT_ROOT'].
            '/dateien/datei.csv','r');

// Inhalt in Variable auslesen

while($c = fgetcvs($fp,1024,',') {
     $zeilen[] .= $c;
}
fclose($fp);
```

4. Häufig verwendete Funktionen

Die hier angerissenen Funktionen werden nur **grob beschrieben**. Wesentlich ausführlichere Beschreibungen und sogar Erfahrungsberichte gibt es auf www.php.net in der Online-Hilfe bzw. in der chm-Hilfe (siehe **Arbeitsmittel**).

Für Einsteiger gilt: Erst lesen, dann ausprobieren und dann nochmal lesen.

strlen

Gibt die Länge einer Zeichenkette zurück.

```
$laenge = strlen($zeichenkette);
```

strpos

Gibt die Position einer Zeichenkette innerhalb einer Zeichenkette zurück. Nicht gefunden ist false, erstes Zeichen ist 0... siehe **Boolsche Operationen**.

```
if(strpos('x=',$string)!==false) echo "Muss Mathe sein";
```

substr

Gibt einen Teil einer Zeichenkette zurück. Kann natürlich auch mit `strpos` kombiniert werden. Parameter 2 ist der Einsteigspunkt und Parameter 3 die Anzahl der zu holenden Zeichen ist.

2 CSV bedeutet „Comma Separated Values“, wobei ein Datensatz eine Zeile ist, und die Werte mit Komma getrennt werden.

```
echo substr(">> Hallo Welt",3,5); // -> "Hallo"
```

is_array

Prüft ob eine Variable ein Array ist.

```
if(is_array($variable)) { echo "Jupp"; }
```

count

Gibt die Länge eines Arrays zurück.

```
if(count($array)>0) { echo "ich arbeite"; }
```

print_r

Gibt ein Array oder ein Objekt rekursiv aus. Es gibt also alle Ebenen in einer relativ übersichtlichen Form aus. Es erzeugt direkt eine Ausgabe. Mit <pre>-Tags sieht es auch im Browser-gut aus.

```
print_r($array);
```

str_replace

Macht Ersetzungen in einer Zeichenkette. Ersetzt Parameter 1 durch Parameter 2 in Parameter 3. Als Parameter 1 und 2 können auch Arrays übergeben werden, wobei diese gleichermaßen iteriert werden und jeweils zusammenpassende Schlüssel dann die zu ersetzenden Werte ergeben.

```
str_replace( 'ö', '&ouml', $string );
```

array_keys, array_values

Funktionen um alle Schlüssel oder alle Werte als Array zu erhalten. Kann z.B. zum sortieren von Arrays oder bei Ersetzungen lustig sein.

```
str_replace( array_keys($arr) , array_values($arr) , $str );
```

preg_replace, preg_match

Mit diesen Funktionen können Reguläre Ausdrücke gesucht oder ersetzt werden. Siehe dazu **Einführung in reguläre Ausdrücke**.

```
if(preg_match('/[^a-zA-Z]/',$str)) {
  echo "Enthält keine Buchstaben";
}
```

explode

Zerlegt eine Zeichenkette anhand eines Trennzeichens. Das Ergebnis ist ein Array der getrennten Zeichenketten.

```
$monate = 'Januar-Februar-März';
print_r( explode('-', $monate) );
// Ergebnis: Array('Januar','Februar','März');
```

implode

Fügt die einzelnen Teile eines Arrays zu einer Zeichenkette zusammen.

```
$teilnehmer = array('Susi', 'Peter', 'Hans');
echo 'Teilnehmer: '. implode(' + ', $teilnehmer) );
// Ergebnis: Teilnehmer:  Susi + Peter + Hans
```

reset
Hiermit wird der interne Position-Zeiger des Arrays auf den Anfang gesetzt. Das macht man z.B. vor einer Iteration mit `while`, `list` und `each`.

```
reset($array);
```

include, require, include_once, require_once

Mit diesen Funktionen können andere PHP Scripte innerhalb des aktuellen Scripts ausgeführt werden. Für Ausführ-Scripte nutzt man i.d.R. `include` und für Funktionssammlungen bzw. Klassen nimmt man `require_once`.

Man sollte immer den vollständigen Pfad zur Datei angeben.

Mit dieser Include-Methode lassen sich ganz einfach Modul-Systeme erstellen.

```
include('ScriptInGleichemVerzeichnis.php');
include($_SERVER['DOCUMENT_ROOT'].'/einscript.php';
require_once($_SERVER['DOCUMENT_ROOT'].'/klasse.php';
if($existiert)
     include('webseite.php');
else
     include('fehler.php');

// Einfaches Modul-System
include($_GET['page'].'.php');
```

asort

Sortiert ein Array anhand der Values, die Schlüssel bleiben dabei am zugehörigen Value erhalten.
Sortiert aufsteigend. Absteigend wäre **arsort**.

```
asort($array); print_r($array);
```

array_push / array_pop / array_shift / array_unshift

`array_push` (hinten-dran) und `array_pop` (hinten-weg) verhalten sich wie ein Stack. `array_shift` (vorne-weg) und `array_unshift` (vorne-dran). Um eine Queue zu simulieren könnte man `array_push` und `array_shift` benutzen.

```
array_push($arr,'neuer Wert');
$wert = array_pop($arr);
$wert = array_shift($arr);
array_unshift($arr,'neuer Wert');
```

unset

Hiermit wird eine Variable aus dem Scope entfernt. Gelöscht sozusagen.

```
unset($variable);
unset($array['key']);
```

htmlentities

Damit Text mit Umlauten die richtige Form hat oder wenn man sicher gehen will, dass in HTML-Eigenschaften keine Gänsefüsschen und überhaupt nur korrekte HTML-Zeichen vorkommen, werden alle Sonderzeichen durch ihr HTML-Äquivalent ersetzt. Also " zu `&quot;`, & zu `&` oder ö zu `&ouml;`

Verwendet man UTF-8, ist die Funktion **htmlspecialchars** besser geeignet, die lediglich <, “ , und > ersetzt.

```
echo '<a name="'. htmlspecialchars('<b>Hallöle "Peter"') .'">';
```

printf

Hiermit können Texte ausgegeben werden und die integrierten Variablen können gleichzeitig formatiert werden. Ein Variablen-Platzhalter beginnt mit % und wird zumindest vom einzubindenden Typ gefolgt. s=String, d=Dezimal, f=Fließkomma.

Soll formatiert werden, kann man zwischen den beiden Zeichen kleine Regeln unterbringen, wie z.B.

`%03d` für eine auf 3 Stellen mit Nullen aufgefüllten Integer.
`%10d` für einen Integer der auf jeden Fall Platz für 10 Zeichen belegt.
`%.2f` für eine auf 2 Nachkommastellen begrenze Fließkommazahl

Die Variablen, die an diese Stellen gesetzt werden werden als Folgeparameter in der Reihenfolge ihres Auftretens in der Zeichenkette gesetzt.

```
printf("Hallo, %s, das kostet $%.2f",$name,$preis);
```

header

Sendet einen HTTP-Header. Dieser wird als Zeichenkette in einer Zeile so angegeben wie er laut HTTP-RFC verlangt wird.

Beispiel Weiterleitung auf andere Seite:

```
header('Location: another-page.php');
exit();
```

Beispiel Angabe des Dokumenttyps:

```
header('Document-type: image/jpeg');
```

Es gibt für alles mögliche eine Funktion in PHP. Die Hilfe gibt über alles Aufschluss.

Außerdem sollte man sich in der PHP-Hilfe über folgende Funktionen schlau machen:

sprintf, split, join, str_repeat, addslashes, stripslashes, trim, assert, die, date, mktime

4.1. Häufige Fehler

Die Fehler unterscheiden sich geringfügig von PHP 3 über PHP 4 zu PHP 5, klingen aber großenteils ähnlich.

4.1.1. Syntaxfehler

`unexpected T_STRING`

Hochkomma/Gänsefüßchen bei String vergessen. U.U. versehentlich PHP-Code (<? php ... ?>)in eine print-Zeichenkette eingebaut.

`unexpected T_ECHO, expecting ',' or ';'`

Strichpunkt am Ende einer Zeile vergessen. Vermutlich die darüber liegende. Bei vielen Strings kann es sich auch um viele Zeilen handeln.

`unexpected $end`

Abschließende } vergessen. Fehler wird am Ende der Seite angemerkt, steckt aber meist mitten im Script, da die letzte schließende Klammer erst am Schluss auffällt.

`unexpected '{'`

Fehler bei Klammersetzung im Schleifen- / Abfragekopf.

4.1.2. Laufzeitfehlverhalten

Benutzt man die verkürzte Schreibweise einer Schleife oder Entscheidung sollte man darauf achten, dass man, falls man selbige um eine Zeile erweitert, nicht vergisst, die { ... } zu setzen.

Die **Einrückung** kann dies verstecken. Man merkt es, wenn eine Zeile ausgeführt wird, obwohl die Entscheidung gar nicht zutrifft.

Es bietet sich an, auf die verkürzte Schreibweise nach Möglichkeit zu verzichten oder sie

immer auf auch in eine Zeile zu schreiben ohne Einrückung.

Bevor man ein Array mit foreach bearbeitet sollte man sicher sein, dass es sich um ein Array handelt, sonst erhält man (sollte das wider Erwarten nicht zutreffen) die Fehlermeldung

```
Invalid argument supplied for foreach().
```

Wird nach komplexen Regeln in mehreren geschachtelten Schleifen mit Zählwerten hantiert, kann es schnell passieren, dass man eine Variable im falschen Schleifenabschnitt zurücksetzt. Dadurch entstehen Endlosschleifen oder sonstiges unvorhergesehenes Verhalten.

Wird eine For-Schleife scheinbar immer nur einmal ausgeführt, so könnte eine Zeile der Form

```
for($i=1;$i<10;$i++); // Strichpunkt
  echo 1;
```

das Problem sein.

Generiert man mit PHP JavaScript kann es passieren, dass **innerhalb des JavaScript Codes** eine PHP-Fehlermeldung ausgegeben wird. Da JavaScript aber nicht angezeigt wird, kann man sie nicht sehen und das einzige was auffällt, ist, dass das JavaScript nicht funktioniert. Werfen Sie immer wieder einen Blick in den HTML-Quelltext der aktuell bearbeiteten Webseite.