Word – Inhaltsverzeichnis 05

1. Ordnen Sie allen roten Texten auf gelbem Hintergrund die Formatvorlage Überschrift 1 zu
2. Ordnen Sie allen grünen Texten auf blauem Hintergrund die Formatvorlage Überschrift 2 zu
3. Ordnen Sie allen Texten in brauner Farbe die Formatvorlage Überschrift 3 zu.
4. Ändern Sie die Formatvorlagen wie folgt:  
   Überschrift 1 22 Pt., Schrift blau, ohne Hintergrundfarbe,  
    Abstand vor/nach: 12 Pt., Seitenumbruch oberhalb  
    (Absatzformatierung)  
   Überschrift 2 16 Pt., Schrift violett, ohne Hintergrundfarbe,  
    Abstand vor: 12 Pt., Abstand nach: 6 Pt.  
   Überschrift 3 14 Pt., braun
5. Erstellen Sie eine neue Formatvorlage mit dem Namen Code. Sie enthält folgende Formatierungen: Consolas 12 Pt.,  
    linker Einzug: 2 cm
6. Ordnen Sie allen Textabschnitten, die php- oder html-Code enthalten die vorher erstellte Formatvorlage Code zu.  
    php-Code beginnt mit <?php und hört mit ?> auf  
    html-Code beginnt mit <html> und hört mit </html> auf
7. Erstellen Sie eine neue Formatvorlage SpezialAufzählung mit dem Symbol ⇨ , einem hängenden Einzug von 0.5 cm und basierend auf der Formatvorlage Standard.
8. Ändern Sie alle Handaufzählungen vom Kapitel Entwicklungsgeschichte von PHP bis und mit dem Kapitel Encoder mit der Formatvorlage SpezialAufzählung.  
   Wahrscheinlich müssen Sie noch einzelne Korrekturen vornehmen
9. Fügen Sie eine Fusszeile ein, die ab der Seite 2 durchnummeriert. Die Seitennummer steht rechtsbündig und hat eine Rahmenlinie oben mit 4 Pt Abstand.
10. Erstellen Sie auf der ersten Seite drei Zeilen unterhalb des Titels ein Inhaltsverzeichnis, das nur die Einträge der Überschriftsebene 1 enthält.

php für Anfänger

**Webscriptsprachen, Methoden**

In der heutigen Webwelt existieren zum Einen serverseitig agierende Scriptsprachen wie zum Beispiel PHP, ASP, JSP. Zum anderen gibt es clientseitig funktionierende Scriptsprachen, deren bekanntester Vertreter wohl JavaScript ist. Serverseitig bedeutet hierbei, dass die Applikation auf dem *Webserver* ausgeführt und deren Ausgabe an den Client (Browser) geschickt wird; clientseitige Applikationen werden direkt auf dem *Client* (also zum Beispiel dem Browser) ausgeführt.

Die Zusammenarbeit zwischen der serverseitigen Scriptsprache und dem Webserver kann hierbei auf unterschiedlichem Weg passieren. Üblich ist zum Beispiel die Verwendung der CGI genannten Technik, bei der der Webserver auf Anfrage einen neuen Prozess mit der jeweiligen Scriptsprache erzeugt. Deutlich ressourcenschonender ist die Verdrahtung der Scriptsprache direkt im Webserver als Modul (bei Apache hat sich hier die Verwendung von PHP als so genanntes DSO, Dynamic Shared Object, eingebürgert).

**Request-Response Prinzip**

1. Client (Browser) schickt die Anfrage an den Webserver

2. Webserver reicht Anfrage an Scriptsprache (zum Beispiel PHP) weiter

3. PHP generiert eine Ausgabe (zum Beispiel HTML-Code)

4. Webserver schickt die Ausgabe an den Client

5. Client stellt die Ausgabe dar

6. Client beendet die Verbindung zum Webserver

**Einbettung in HTML-Code**

PHP-Scripte können direkt in den HTML-Code eingebettet werden, wie folgendes Beispiel zeigt.

<html>

<body>

<?php

print “Hallo<br>\n”;

?>

</body>

</html>

**Typische Dateiendungen**

Damit der Webserver weiß, welche Scriptsprache gestartet werden muß, wird anhand der Dateiendung der vom Client angeforderten Seite (z.B. http://www.thinkphp.de/index.php ) festgelegt, ob zum Beispiel PHP für Webseiten mit der Dateiendung *.php* gestartet wird.

Übliche Dateiendungen sind hierbei: .phtml (ganz alt), .php3 oder .php

.php ist die von der PHP Group empfohlene Dateiendung.

**Geschichte von PHP**

PHP ist noch eine sehr junge Scriptsprache, wie die folgende chronologische Auflistung zeigt.

**Entwicklungsgeschichte von PHP**

- Ausgedacht im Herbst 1994 von Rasmus Lerdorf

- PHP Version 1 im Frühling 1995

- PHP Version 2 (PHP/FI1 genannt) von 1995 bis 1997

- PHP Version 3 (kompletter Rewrite) von 1997 bis 2000

- PHP Version 4 (neuer Meilenstein) im 2. Quartal 2000

- PHP auf dem Weg zur universellen Scriptsprache? PHP-GTK2 Extension im 1. Quartal 2001, darüber hinaus readline- und ncurses-Extensions für die Shell

- In naher Zukunft: Application Server oder Ressourcen Manager? (http://www.vl-srm.net/)

**Entwickler-Team von PHP**

Ca. 400 Leute mit CVS write access, davon ca. 40 aktiv an der Entwicklung von PHP beteiligt. Viele Dokumentations- und Übersetzungsteams, Quality Assurance Team (http://qa.php.net/), PEAR Development Team (http://pear.php.net/)

- Zeev Suraski und Andi Gutmans in Isreal

- Shane Caraveo in Florida

- Stig S. Bakken in Norwegen

- Andrei Zmievski in Lincoln, Nebraska

- Sascha Schumann und Thies C. Arntzen 3in Deutschland

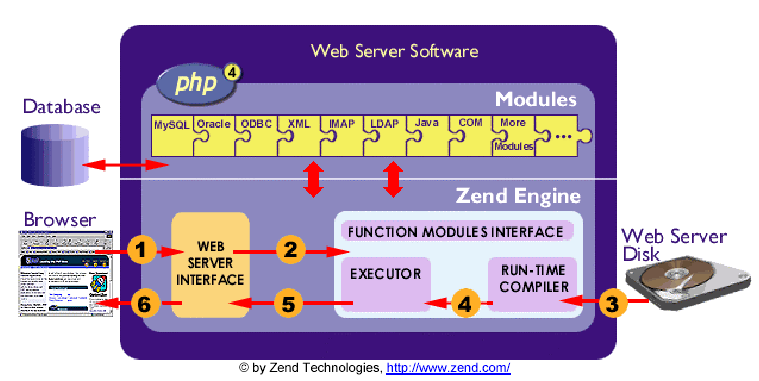
- Jim Winstead in LA

- Sam Ruby in Raleigh, NC

- Rasmus Lerdorf in San Francisco

**Aufbau von PHP4**

Seit PHP3 hat sich der Aufbau von PHP grundlegend geändert. Die Zend Engine (die mittlerweile unter einer BSD-ähnlichen Lizenz steht) ist der Kern von PHP. Sie ermöglicht ein Interface zu den Funktionsmodulen (Extensions), das eigentliche Ausführen der PHP-Scripte sowie einen Run-Time Compiler. Die ZendEngine interagiert über die SAPI (Server API) mit dem Web Server.



Die ZendEngine kann selbst durch eigene Extensions erweitert werden. Beispiele sind die hierfür von Zend Technologies erhältlichen kommerziellen Tools wie der ZendCache (Beschleunigen von PHP Applikationen) und der ZendEncoder (Verschlüsseln von PHP Sourcecode) sowie frei verfügbare Extensions/Tools wie der PHP Debugger DBG (http://dd.cron.ru/dbg/), der APC Cache (http://apc.communityconnect.com/) und die PHPDoc Extension.

**Wichtige Informationsquellen, Teil 1**

Damit der PHP Entwickler seine Arbeit wesentlich effizienter und leichter durchführen kann, gibt es verschiedene Websites, Tools und Bücher, die man kennen sollte. Einiges davon listen wir hier auf.

**Websites**

- <http://www.php.net> (die Mutter aller PHP Websites. )

- <http://www.zend.com/> (Heimatseite von Zend Technologies Ltd., Anbieter von kommerziellen Tools wie ZendCache, ZendEncoder und ZendIDE, qualitativ hochwertige Artikel über das PHP Development, weekly summaries der PHP Developer Mailingliste.)

- <http://www.zugeschaut-und-mitgebaut.de/php/> (vollständige Funktionstabelle. Hier werden auch die undokumentierten PHP Funktionen aufgelistet.)

- http://www.php4win.com/ (PHP Builds für Windows. Andreas Otto und Daniel Beulshausen stellen mittlerweile offiziell die Windows Builds von PHP zusammen.)

**Nützliche Tools**

- PHPLIB <http://phplib.sourceforge.net/>

- phpMyAdmin <http://sourceforge.net/projects/phpmyadmin/>

- phpOracleAdmin <http://www.phporacleadmin.org/>

- PHPDoc <http://www.phpdoc.de/>

- Menu3 <http://www.ulf-wendel.de/projekte/menu/>

**Newsgruppen, Mailinglisten, IRC**

- <http://www.phpcenter.de/php-de/> (deutschsprachige Mailingliste mit ca. 1600 Teilnehmern)

- <news:de.comp.lang.php> (deutschsprachige Newsgruppe, mittleres Niveau)

- <http://php.net/support.php> (Verschiedene englischsprachige Mailinglisten, auch als Newsgruppe benutzbar. Auf PHP-DEV diskutieren die PHP Core Developer, PHP-General ist für allgemeine Fragen/Probleme von PHP vorhanden.)

- IRCNet #php.de, EFNet #php: auf #php.de findet man regelmäßig die aktiven deutschsprachigen PHP’ler, auf #php sind auch die Core Developer Rasmus, Zeev und Co. zu finden.

**Deutschsprachige Websites**

- http://www.phpcenter.de/

- http://www.dynamic-webpages.de/

- http://www.php-homepage.de/

**Wichtige Informationsquellen, Teil 2**

Nicht nur online, sondern auch offline sind mittlerweile viele Informationen in Form von Büchern und ab 2002 auch in Form eines PHP Magazins verfügbar.

**Bücher**

- verschiedene aufgelistet u.a. auf http://www.phpcenter.de/ und http://www.php.net/books.php

- Tipp für Anfänger: „PHP & MySQL“ (Data Becker)

- Tipp für Fortgeschrittene: „Web Application Development with PHP4.0“

- Wer gerne Funktionsreferenzen in Papierform hat: “PHP functions. Essential Reference.“ von Zak Greant

- Die zukünftige Referenz: “Programming PHP” aus dem O’Reilly Verlag

**Magazine**

Ab 2002 erscheint beim Software & Support Verlag das (deutschsprachige) PHP Magazin, das sich an professionelle PHP Programmierer wendet. Mehr Informationen findet man auf http://www.phpmag.de/

**Produkte/Applikationen um PHP**

Die Einführung von kommerziellen Tools wie ZendCache und die weite Verbreitung von PHP an sich zog das Interesse der OpenSource Community nach sich, auch frei verfügbare Applikationen/Tools für PHP anzubieten. Die folgende, nicht vollständige Aufstellung soll zeigen, was momentan auf dem Markt verfügbar ist:

**Debugger**

- ZendDebugger als Bestandteil der ZendIDE, kommerziell

- Der zur Zeit am weitesten fortgeschrittene Debugger, **DBG** von Dmitri Dmitrienko, frei verfügbar, inklusive SE IDE (Windows) mit Profiling (http://dd.cron.ru/dbg/)

- Der kleinere Bruder des APC Cache, **APD**, frei verfügbar

- Die Entwicklungsumgebung Komodo von ActiveState kommt ebenfalls mit einem eigenen Debugger daher

**Caches**

- **ZendAccelerator**, kommerziell

- APC Cache, frei verfügbar (http://apc.communityconnect.com/)

- Bware Cache, frei verfügbar, auch unter Windows einsetzbar

**Encoder**

- ZendEncoder, kommerziell

- **PBC PHP Bytecode Compiler**, frei verfügbar (http://pbc.sourceforge.net/)

**Entwicklungsumgebungen**

Derlei gibt es viele, eine Auflistung aller wäre garantiert unvollständig. Genannt werden sollten XEmacs, NuSphere PHPEd Advantage (kommerziell), Magumas PHP4EE (kommerziell), PHP IDE (http://www.phpide.de/), Zends ZendIDE und unzählige andere. Linux-Benutzer schwören auf Editoren wie zum Beispiel XEmacs, (g)vim, Quanta oder KDevelop.

**Einbindung in HTML**

Es gibt vier Möglichkeiten zur Einbindung von PHP in HTML-Dokumente:

- <? phpcode ?>

- <?php phpcode ?>

- <% phpcode %>

- <script language= »php »> code</script>

Die zweite Variante ist die am meisten unterstützte und portabelste Variante, um PHP einzubetten. Sie wird insbesondere dann gebraucht, wenn man XML-konforme Dokumente schreiben möchte.

**Beachten** sollte man, sich auf einen Stil zu einigen: es macht keinen Sinn und trägt nur zur Unübersichtlichkeit bei, wenn man innerhalb eines Projektes oder, noch viel schlimmer, innerhalb eines PHP-Scripts alle vier verschiedenen Varianten benutzt.

Desweiteren gibt es eine Kurzschreibweise für „print $variable“, wie in ASP:

- <?= $variable; ?>

- <%= $variable; ?>

Diese Flexibilität und Einfachheit von PHP, den Programmcode innerhalb eines HTML-Dokuments einbetten zu können (bzw. über print auch HTML-Code auszugeben), ist zugleich auch ein großer Nachteil: gerade in größeren Projekten, in denen Webdesigner und Programmierer zusammenarbeiten müssen, ist es sinnvoller, über Template-Mechanismen den Programm- vom HTML-Code zu trennen. Geeignet ist diese Methode jedoch erst für den Fortgeschrittenen PHP-Programmierer. Der Anfänger sollte sich zunächst mit der Einbettung in HTML begnügen, um mit PHP vertraut und sicherer zu werden.

<html>

<head>

<title>Dies ist ein PHP-Script</title>

</head>

<body bgcolor=”#ffffff”>

<?php

print “Ich bin ein toller Text.”;

print „<br>und ich kann auch <b>**HTML**</b> enthalten<br>“;

?>

</body>

</html>

(Beispiel der Einbettung von PHP in den HTML-Quelltext)

**Kommentare in PHP**

Seinen Quelltext zu kommentieren ist sehr wichtig, gerade bei größeren Projekten. Wie oft passiert es, dass man mehrere Wochen nicht an einem Projekt weiterarbeiten kann und dann plötzlich feststellt, dass man seinen eigenen Programmcode nicht mehr versteht und sich wieder mühsam und zeitintensiv (und damit auch kostenintensiv) einarbeiten muß.

PHP kennt verschiedene Methoden der Kommentierung:

<?php

print „foo“; # einzeilig, Bashshell-artig

print „foo“; // einzeilig, C++-artig

/\* ein- oder mehrzeilig

\*/

print „foo“;

?>

Wichtig ist, dass Sie sich für einen Kommentierungsstil entscheiden. Dies erhöht die Übersichtlichkeit. Empfehlenswert ist eine Kommentierung nach dem PHPDoc Standard (http://www.phpdoc.de/), um ggf. automatisch eine handliche Sourcedokumentation zu erhalten.

<?php

/\*\*

\* this function will print the script name

\*

\* it will happen that you have a longer description of what a function

\* does. this is nice

\*

\* @PARAM string new parameter

\* @return Boolean if it was true or not

\*/

function print\_myself($str) {

print $GLOBALS[“PHP\_SELF”];

return TRUE;

}

?>

**Variablen**

Variablen sind essenzielle Elemente einer Programmiersprache. In ihr speichern Sie Daten unterschiedlichen Typs. Stellen Sie sich eine Variable wie einen Schuhkarton vor: er hat eine Beschriftung (den Variablennamen) sowie einen bestimmten Inhalt (der Wert, der der Variablen zugewiesen wurde).

Variablen in PHP beginnen mit einem **$**, gefolgt vom Variablennamen. Für den Variablennamen gelten dabei folgende Vorschriften:

• Namen unterliegen den gleichen Regeln wie die der anderen Bezeichner in PHP

• Der Name beginnt mit einem Unterstrich (**\_**) oder mit einem Buchstaben

• ... gefolgt von einer beliebigen Anzahl von Buchstaben, Zahlen oder Unterstrichen

Eine Liste von vordefinierten Variablen (z.B. **$PHP\_SELF**, die den Namen des eigenen Scriptes enthält) erhält man durch Gebrauch von <? phpinfo() ?>

<?php

$var = "foo";

$\_\_hallo = "blubb";

echo $\_\_hallo; // gibt "blubb" aus

?>

**Typen**

Um den Inhalt einer Variablen zu klassifizieren, gibt es verschiedene Typen. Man unterscheidet hierbei zwischen skalaren und zusammengesetzten Typen.

Skalare Typen sind u.a.:

• String (Zeichenkette)

• Integer (ganzzahliger Wert)

• Float (Gleitkommazahl)

• Boolean (TRUE, FALSE)

Zusammengesetzte Typen:

• Array

• Object (Objektorientierte Programmierung)

Daneben gibt es seit PHP4 noch weitere Typen: NULL sowie ResourceID (z.B. Return-Wert von mysql\_connect() ).

**Skalare Variablentypen**

**Integer**

<?php

/\* Integer Wert \*/

$myint = 50;

$hexwert = 0xffffff;

print „Hexwert: $hexwert“;

/\* Oktale Notation (Basis 8) \*/

$oktal = 013;

print „Oktalwert: $oktal“;

?>

**Float**

<?php

$float = -0.2345;

$float\_plus = 24.345;

?>

**String**

<?php

$zeichenkette = “Hallo, ich bin ein Text”;

?>

**Boolean**

<?php

$bool = TRUE;

$nein = FALSE;

?>

**Zusammengesetzte Variablentypen**

**Arrays**

Erinnern wir uns an das Beispiel mit dem Schuhkarton zurück: ein Array ist prinzipiell auch ein Schuhkarton mit einer Beschriftung, allerdings kann der Schuhkarton beliebig viele Elemente und wiederum beliebig viele andere Schuhkartons (mehrdimensionales Array) enthalten.

Arrayelemente bestehen aus einem Key (zur Identifizierung des einzelnen Elements) und dem entsprechenden Value (Inhalt). Bei indizierten Arrays ist der Key numerisch (idR beginnend mit 0). Assoziative Arrays (hash tables) können beliebige Datentypen als Key haben (üblich sind Strings).

<?php

/\* Initialisierung eines Arrays. \*/

$myarr = array();

$myarr[] = 5; // Anfügen eines Elements, Key Index wird automatisch erzeugt (hier: 0)

$myarr[] = „string“;

$myarr[2] = „foo“; // Direkte Zuweisung eines Array-Elements über den Index 2

print\_r($myarr);

/\* Assoziatives Array \*/

$neues\_array = array(“key” => “value”);

print\_r($myarr);

/\* mehrdimensionales, assoziatives Array \*/

$multi = array(

“key“ => “value”,

“bgcolor“ => 5,

“dbhost” => “localhost”,

“nocheins” => array(0,15,20,25),

15 => array(

“key2” => “bla”,

“key3” => 0

)

);

print\_r($multi) ;

?>

**Objekte**

PHP kennt eine einfache objektorientierte Programmierung, die durch Verwendung der neuen ZendEngine2 in PHP5 sehr stark erweitert wird (Dekonstruktoren etc.). Prinzipiell sind Objekte nichts anderes als eine Sammlung von Methoden (= Funktionen) sowie Eigenschaften (= Daten). Grundsätzlich können Objekte auch vererbt werden.

**Vordefinierte Variablen**

<? phpinfo(); ?>

Mit diesem Mini-Script erhalten Sie alle in PHP vordefinierten Variablen (und weitere Infos, z.B. über die in PHP eincompilierten Module). Wichtige vordefinierte Variablen sind:

• $PHP\_SELF: Name des eigenen Scriptes

• $GLOBALS: ein assoziatives Array, das alle globalen Variablen enthält. ( print\_r($GLOBALS);, um den Inhalt anzuschauen)

• $argc, $argv: bietet Zugriff auf Kommandozeilenparameter (hierfür muß PHP als CGI Binary compiliert worden sein)

• $php\_errormsg: enthält die zuletzt gelieferte Fehlermeldung („track\_errors = On“ muß hierfür in der php.ini stehen)

<?php

print\_r($GLOBALS);

$vordefiniert = „Hallo“;

print $vordefiniert;

print „<br>“;

print $GLOBALS[„vordefiniert“]; (kein $ bei „vordefiniert“ !)

print „<a href=’$PHP\_SELF’>Neu laden?</a>“;

?>

Weitere assoziative Arrays sind:

• $HTTP\_POST\_VARS: alle Variablen, die über ein Formular geliefert werden

• $HTTP\_GET\_VARS: alle Variablen aus GET (Browser-URL)

• $HTTP\_COOKIE\_VARS: alle Cookie-Variablen

Ab PHP 4.1.0 existieren weitere Variablen, die als Ersatz für die $HTTP\_\*\_VARS geplant sind. Ihr Vorteil: sie existieren in jedem Scope (d.h. automatisch auch in Funktionen/Methoden), sind in ihrer Schreibweise viel kürzer und sollen die Entwickler dazu animieren, über sicheres Programmieren (siehe die entsprechende Schulung von ThinkPHP) nachzudenken und sorgfältig mit PHP-Scripten umzugehen:

• $\_GET

• $\_POST

• $\_COOKIE

• $\_SERVER: enthält alle Servervariablen (z.B. REMOTE\_ADDR für die IP-Adresse des Clients)

• $\_ENV: enthält Environment-Variablen

• $\_REQUEST : eine Mischung der GET-, POST- und Cookie-Variablen

• $\_SESSION: enthält die im PHP4 Session System registrierten Variablen

Weitere Infos hierzu finden Sie auf http://www.php.net/release\_4\_1\_0.php

Übung:

Schreiben Sie ein PHP-Script, das Ihnen die Inhalte der globalen Variablen **$HTTP\_POST\_VARS**, **$HTTP\_GET\_VARS**, **$HTTP\_COOKIE\_VARS** mit Hilfe der Funktion print\_r() ausgibt. Vergleichen Sie dabei diese Inhalte mit denen, die ein print\_r($GLOBALS); liefert.

**Variable Variablen**

Variable Variablen (Zungenbrecher) dienen dazu, um den Inhalt einer Stringvariablen als Variablennamen zu interpretieren. Sie können damit, wenn die Stringvariable zum Beispiel aus einer externen Quelle kommt, auf ihrer Applikation namentlich nicht bekannte Variablen zugreifen.

Zwei Notationen sind möglich: $$name oder **${$name}**, wobei die letztere Variante bevorzugt werden sollte, da hieraus klar hervorgeht, welche Variable Sie ansprechen möchten.

<?php

$var = „Inhalt“;

$var2 = „var“;

print $$var2; /\* Ausgabe: Inhalt \*/

print ${$var2};

${$var2} = „foo“;

print $var;

?>

Problem: variable Variablen erzeugen unübersichtlichen Code. Sie sollten daher nur dann gebraucht werden, wenn es *wirklich* nötig ist.

**Konstanten**

Konstanten sind auch schon aus anderen Programmiersprachen (z.B. C) bekannt. In PHP können sie nur skalare Werte annehmen. Konstanten werden meist zur Konfiguration der Applikation benutzt (z.B. Datenbankparameter etc.) und sollten in einer zentralen Datei definiert werden.

Mit der Funktion defined() kann überprüft werden, ob eine Konstante bereits definiert ist.

<?php

define(“DB\_HOST“,“localhost“);

define(“DB\_USER”,”root”);

define(“DB\_PASS”,””);

mysql\_connect(DB\_HOST, DB\_USER, DB\_PASS);

if (defined(“DB\_HOST”)) { print “DB\_HOST is defined”; }

?>

Wie zu sehen ist, beginnen Konstanten nicht mit einem $, sie sind case sensitive (d.h. Groß- und Kleinschreibung wird unterschieden) und werden innerhalb von Strings (siehe das print-Statement) **nicht** interpretiert. Des weiteren gelten für Konstanten die gleichen Namensbenennungsregeln wie für Variablen.

Die case-Sensitivität kann über einen 3. Parameter von define() außer Kraft geschaltet werden:

<?php

define(“DB\_HOST“, “LOCALHOST“, TRUE);

print DB\_HOST;

print “<BR>”;

print db\_host;

?>

**Operatoren**

PHP kennt verschiedene Operatoren:

• vollständige Liste auf http://www.php.net/manual/en/language.operators.php

• +, -, \*, /, %

• . als Concatenator bei Strings (in PHP/FI "+")

• C-like: +=, %=, -=, \*= etc. (z.B. $a += 5;)

• = (Zuweisung), == (Wertgleichheit), === (Wert- und Typgleichheit, seit PHP4)

• @ zur Unterdrückung von Fehlermeldungen. Die Variable $php\_errormsg enthält die Fehlermeldung und ermöglicht somit CI konforme und nutzerfreundliche Fehlerbehandlung.

• && , AND

• || , OR

<?php

$foo = 5;

$bar = 10;

if ($foo == $bar) { print “Wertgleich“; }

if ($foo === $bar) { print “Wert- und typgleich”; }

$fname = „/tmp/irgendwas“;

if (!$fp = @fopen($fname,“r“)) {

print “Fehlermeldung: $php\_errormsg”;

}

?>

**Kontrollstrukturen**

Applikationen durchlaufen in der Regel nicht nur einen Weg, sondern können individuelle Wege gehen. Damit dies möglich ist, benötigt man Ablaufkontrollmechanismen. Folgende Kontrollstrukturen kennt PHP. Beachten Sie, dass die in den Beispielen geschriebenen geschweiften Klammern weggelassen werden können, wenn der Kontrollstruktur nur ein Statement folgt.

• if

• else

• elseif

• while

• do

• for

• foreach

• break

• continue

• switch

• require / require\_once

• include / include\_once

**if/else/elseif**

Einfache Abfrageverzweigungen. Verschachtelungen sind ebenfalls möglich.

<?php

$foo = 5;

if ($foo == 3) {

// wenn $foo gleich 3, dann führe diesen Block aus

} elseif ($foo == 4) {

// andernfalls, wenn $foo gleich 4, dann führe diesen aus

if ($bar == „hallo“) {

}

} else {

// wenn keine der vorherigen Bedingungen zutrifft.

}

if ( $foo == 3 AND $bar == 5) {

// Verkettung mehrerer Bedingungen möglich

}

?>

**Vorsicht!** Oft passiert es, dass man schreibt: if ($foo = 3) ! Damit wird eine Variablenzuweisung gemacht, da PHP den Ausdruck innerhalb des if-Statements interpretiert. Um das Debugging zu erleichtern, ist es klug, folgendermaßen zu schreiben:

<?php

if (3 == $foo) {

}

?>

Wenn man das eine “=” vergißt, so wird das Debuggen einfacher, da PHP einen Syntax Error als Fehlermeldung wirft.

**while-Schleifen**

while (expr) {

}

do {

} while (expr);

while-Schleifen werden ausgeführt, solange der Ausdruck expr wahr ist. Bei der letzten, fußgesteuerten Variante wird die expression am Ende der Schleife überprüft.

<?php

$i = 0;

print „while-Methode:<br>“;

while ($i < 3) {

$i++; // $i um eins erhöhen

print „i = $i<br>“;

}

print „<hr>do...while Methode:<br>”;

do {

$I++;

print “i = $i<br>”;

} while ($i < 3);

?>

Ausgabe:

while-Methode:

i = 1

i = 2

i = 3

do..while Methode:

i = 4

**for-Schleifen**

for-Schleifen sind die komplexesten Schleifen in PHP. Ihre Syntax ist:

for (expr1; expr2; expr3) { }

• expr1 wird bei Schleifenbeginn ausgeführt

• zu Beginn jeden Durchlaufs wird expr2 überprüft

• ist expr2 wahr, wird die Schleife ausgeführt

• am Ende jedes Durchlaufs wird expr3 ausgeführt

• jeder Parameter kann leer sein; ist expr2 leer, wird Schleife unbestimmt oft durchlaufen, da PHP ihn als TRUE wertet.

<?php

for ($i=0; $i<=10; $i++) {

print „Schleife, i = $i<br>“;

}

?>

**foreach, break/continue**

Seit PHP4 existiert die foreach-Schleife zum Durchlaufen von Arrays. Dabei arbeitet foreach mit einer Kopie des Arrays (und sollte deswegen bei sehr sehr großen Arrays nicht verwendet werden).

<?php

$arr = array(

„key1“ => „var1“,

„dbhost“ => „localhost“,

13 => “hallo”

);

// key und value jedes einzelnen Elements werden in $k resp.

// $v kopiert

foreach ($arr as $k => $v) {

print “$k => $v<br>”;

}

// hier wird nur die value jedes Elements kopiert.

foreach ($arr as $v) {

Print “Value = $v<br>”;

}

?>

**break/continue**

Mit break kann der Ablauf einer Schleife unterbrochen werden. Mit continue können Sie direkt den nächsten Schleifendurchlauf erzwingen.

<?php

for ($i=0; $i<=10; $i++) {

print “i = $i<br>”;

if ($i == 3) continue;

if ($i == 5) break;

print “Nochmal: $i<br>”;

}

print “Ende”;

?>

Ausgabe:

i = 0

Nochmal: 0

i = 1

Nochmal: 1

i = 2

Nochmal: 2

i = 3

i = 4

Nochmal: 4

i = 5

Ende

**switch/case-Anweisungen**

Oft möchte man ein- und dieselbe Variable mehrmals vergleichen. Da kann eine if-Abfrage schon sehr unübersichtlich wirken. Mit einer switch-Anweisung können Sie dies übersichtlicher gestalten:

<?php

$i = 3;

switch ($i) {

case „hallo“:

print “I heißt hallo”;

break; // diesen case-Teil beenden.

case 3:

print „i hat den Wert 3“;

break;

case 50:

print „i hat den Wert 50“;

break;

default:

// wenn die vorherigen Fälle nicht zutreffen.

break;

}

?>

• case: Wert für den zu prüfenden Ausdruck

• break; unterbricht die Ausführung der aktuellen switch Anweisungs-Sequenz

• default: Schlüsselwort: falls die case-Anweisungen nicht zutreffen

**include/require**

In der Regel wird Ihr Programmcode nicht in einer einzigen Datei stehen, sondern übersichtlich auf mehrere Dateien aufgeteilt werden. Damit ein Script Programmteile aus einer anderen Datei einbinden kann, gibt es include, require bzw. die Pendants include\_once und require\_once.

**Bitte beachten Sie:** das Includen/Requiren eines PHP-Scripts ist **nicht** gleichzusetzen mit einem Aufruf im Browser plus Parameterübergabe! Ein

include „foobar.php?param=5”;

ist also nicht möglich.

Unterschied zwischen include und require: include liest die angegebene Datei ein und interpretiert sie. Require stellt den Sourcecode der einzubindenden Datei an die Stelle der require-Anweisung.

Die entsprechenden \*\_once-Pendants sorgen dafür, dass die angegebene Datei nur ein einziges Mal eingebunden wird. Die Überprüfung erfolgt anhand einer internen Hash-Tabelle, in der festgehalten wird, welche Dateien bereits eingebunden wurden.

Bei den einzubindenden Dateien gelten die gleichen Regeln wie bei normalen PHP-Scripten, d.h. sie müssen mit dem Opening-Tag eingeleitet und mit ?> beendet werden. Eine Mischung zwischen HTML- und PHP-Code ist ebenfalls möglich.

Datei foo.php:

<?php

print $a;

?>

Datei index.php:

<?php

$a=5;

include „foo.php“;

?>

Ausgabe:

5

**Ausführlicher: Arrays**

Um die Mächtigkeit von Arrays zu bestimmen, kennt PHP drei verschiedene Wege:

|  |  |  |
| --- | --- | --- |
| count/for | foreach | Reset/while/list/each |
| Nur geeignet für fortlaufend numerisch indizierte Arrays. | Erst seit PHP4 vorhanden, arbeitet auf einer Kopie des Arrays. Bei kleinen Arrays die schnellste und komfortabelste Variante. | Funktioniert mit PHP3 und PHP4 sowie assoziativen Arrays. |

Beispiel: Arrays auslesen.

<?php

$num = array(0,10,15,20,35,23);

$mixed = array(

“dbhost” => “localhost”,

“dbuser” => “root”,

“dbpass” => “”,

15 => “test”

);

// count ermittelt die Anzahl der Elemente eines Arrays.

for ($i=0; $i<count($num); $i++) {

print “num[$i] = “.$num[$i].”<br>”;

}

foreach ($mixed as $k => $v) {

print “$k => $v<br>”;

}

reset($mixed);

while (list($k,$v) = each($mixed)) {

print “$k => $v<br>”;

}

?>

**Array-Funktionen**

Es gibt weitere nützliche Array-Funktionen, die seit PHP4 hinzugekommen sind:

• var\_dump (bereits seit PHP >= 3.0.5)

• print\_r

• array\_pop (Entfernen und Zurückliefern des letzten Elements)

• array\_push (Hinzufügen eines oder mehrerer Elemente)

• array\_slice (Zurückliefern bestimmter Elemente eines Arrays)

• array\_merge (Zusammenführen von Arrays)

• array\_merge\_recursive (rekursives Zusammenführen)

• in\_array

Sortieren von Arrays:

• asort (Sortieren der Values)

• arsort (reverses Sortieren)

• ksort (Sortieren der Keys/Indizes)

• krsort (reverses Sortieren der Keys/Indizes)

• Benutzerdefiniert: uasort, uksort, usort

Alle Array-Funktionen finden sich im PHP Manual an geeigneter Stelle und gehören zum Pflichtstudium jedes PHP Softwareentwicklers.

<?php

$arr = array(0,15,56);

array\_push($arr, 25, array(0,25,3), 67);

print\_r($arr);

?>

**Funktionen**

Funktionen dienen dazu, Programmteile in logische Einheiten zu gruppieren. Wenn man zum Beispiel ein und denselben Programmcode mehrmals aufrufen muß, so sind Funktionen sehr gut dafür geeignet.

• Funktionen haben per default keinen returnwert, können jedoch einen beliebigen Wert per *return* zurückgeben.

• eigener Namensraum

• optionale Parameter ( $bla,$foo="bar" )

• variable Anzahl von Parametern ab PHP4, Hilfsfunktionen func\_num\_args() und func\_get\_args()

• Funktionsschachtelungen möglich

• es gibt keine überladenen Funktionen

• create\_function() zur on-the-fly Erzeugung von Funktionen

• Rekursion möglich

• Argumentübergabe auch als Referenz

• variable Funktionen

Eigener Namensraum: in Funktionen stehen globale Variablen nicht automatisch zur Verfügung. Man kann auf diese jedoch über das assoziative Array $GLOBALS zurückgreifen oder die Variable der Funktion mit Hilfe des **global** Statements bekannt machen:

<?php

function foo() {

print „foo hier. A = $a<br>”; // funktioniert nicht.

}

function bar() {

print „bar hier. A = “.$GLOBALS[“a”].”<br>”; // funktioniert.

}

function bar2() {

global $a;

print “bar2 hier. A = $a<br>”; // funktioniert.

}

$a=5;

foo();

bar();

bar2();

?>

Ausgabe:

foo hier. A =

bar hier. A = 5

bar2 hier. A = 5

**Funktionen: Beispiele**

<?php

// Beispiel einer Funktion mit einem default-Parameter.

function foo($param1, $param2=5) {

print “foo. param1 = $param1, param2 = $param2<br>”;

}

foo(5);

foo(5,10);

function returnwert($a) {

print “returnwert. a = $a<br>”;

return $a+2;

}

$a=2;

$a=returnwert($a);

print “draussen. a = $a<br>”;

function variabel() {

$anz\_args = func\_num\_args();

$arg\_liste = func\_get\_args();

for ($i=0; $i<$anz\_args; $i++) {

print $arg\_liste[$i].”<br>”;

}

}

variabel(1,”hallo”,5);

variabel(2,5,”string”,”nocheiner”);

?>

Ausgabe:

foo. param1 = 5, param2 = 5

foo. param1 = 5, param2 = 10 returnwert. a = 2

draussen. a = 4

1

hallo

5

2

5

string

nocheiner

**Formulare in PHP**

Eine der am meisten verwendeten Mechanismen zur Interaktion mit dem Benutzer sind HTML-Formulare. Die mit dem name= Parameter gesetzten Namen für Formularelemente stehen nach dem Submit als PHP-Variablen mit den vom Benutzer eingegebenen Daten zur Verfügung.

Eingebürgert hat sich die Verwendung von so genannten Affenformularen. Dabei ruft das Script, das das Formular darstellt, sich selbst wieder auf. Die Unterscheidung, ob nun das Formular dargestellt werden soll oder ob es übermittelt wurde, wird durch Prüfung von isset() gefällt:

<?php

if (isset($submit)) {

print “Text: $txt”;

} else {

// Noch nicht vorhanden -> Formular darstellen.

?>

<form action=“<?php print $PHP\_SELF; ?>” method=”POST”>

Text: <input type=”text” name=”txt”><br>

<input type=”submit” name=”submit” value=”Abschicken”>

</form>

<?php

}

?>

Enthält das Formular Werte, die nicht in der URL angezeigt werden sollen und die ggf. nicht Bestandteil des Referers sowie nicht in Proxy-Logs auftauchen sollen, so sollte man die POST-Methode verwenden. Dies ist insbesondere dann der Fall, wenn ein **password** Eingabefeld verwendet wird.

Ebenfalls anzuraten ist die POST-Methode, wenn TEXTAREA Elemente im Formular vorkommen (bei denen die Länge der Eingabe nicht begrenzt ist), da eine GET-URL nur eine bestimmte Anzahl an Zeichen (in der Regel 2048) lang sein darf.

Zwingend notwendig wird die POST-Methode, wenn man ein Fileupload-Element in seinem Formular hat. Der notwendige enctype="multipart/form-data" Parameter im <form>-Tag setzt die POST-Methode zwingend voraus.

**select, checkbox Formularelement**

<select multiple> bzw. <select> Elemente werden wie folgt verarbeitet:

<form action="script.php">

<select multiple size=3 name="myvar[]">

<option value="1">Eins

<option value="2">Zwei

<option value="3">Drei

</select>

<br>

<input type="submit" name="abgeschickt" value="Go!">

</form>

Wichtig und entscheidend ist, dass der Name der Variablen im select-Tag mit eckigen Klammern endet, damit ein Array erzeugt wird. Sollte ein normales select Element erzeugt und verarbeitet werden, bei dem der Benutzer nur eine Element auswählen kann, so sind die eckigen Klammern nicht nötig. Das Script script.php erhält in der Variablen $myvar ein Array mit den ausgewählten Elementen. Ein var\_dump($myvar); bzw. print $myvar; (beim einfachen Select) zeigt, welche Elemente ausgewählt wurden.

Beim select Multiple sendet der Browser im Request folgende Daten mit, wenn alle drei Elemente ausgewählt wurden:

myvar[]=1&myvar[]=2&myvar[]=3

PHP macht hieraus ein :

$myvar[]=1;

$myvar[]=2;

$myvar[]=3;

… und schon wird klar, warum hier eckige Klammern benötigt werden, um ein Array zu erzeugen.

**Checkbox**

Sind Checkboxen nicht markiert, werden sie gar nicht übermittelt. Andernfalls haben sie den im value= angegebenen Wert.

<input type="checkbox" name="box[]" value="yes">

<input type="checkbox" name="box[]" value="yes">

if (isset($box)) {

var\_dump($box);

} else {

print "Es wurde nichts angeklickt.";

}

**Radiobuttons**

Radiobuttons funktionieren ähnlich wie Checkbox-Elemente. Wird ein Radio-Button angewählt, so wird sein Wert übermittelt. Wird der Radio-Button nicht angewählt, so wird die Variable nicht an PHP übermittelt, d.h. ein isset($variablenname) würde FALSE liefern.

Gruppen von Radio-Buttons erzeugt man, indem man den Buttons jeweils den gleichen Namen gibt.

<?php

if (isset($submit)) {

if (isset($eins)) {

print “eins = $eins<br>“;

} else {

print „\$eins ist nicht gesetzt“;

}

if (isset($gruppe)) {

print „gruppe = $gruppe<br>“;

} else {

print „\$gruppe ist nicht gesetzt”;

}

} else {

?>

<form action=“<?php print $PHP\_SELF; ?>” method=”post”>

<input type=”radio” name=”eins” value=”CDs”>CDs<br><br>

<input type=”radio” name=”gruppe” value=”DVDs” checked>DVDs<br>

<input type=”radio” name=”gruppe” value=”Boxen”>Boxen<br>

<br><input type=”submit” name=”submit” value=”OK”>

</form>

<?php

}

?>

**Textarea**

Eine Textarea wird in HTML anders definiert als zum Beispiel ein Texteingabefeld.

<form action=“<?php print $PHP\_SELF; ?>” method=”POST”>

<textarea name=”feldname” cols=”30” rows=”5”><?php

print htmlspecialchars($feldname); ?></textarea>

</form>

Soll bereits Text in der Textarea angezeigt werden, so steht dieser zwischen dem öffnenden und dem schließenden textarea-Tag. Hierbei wird die PHP-Funktion *htmlspecialchars()* verwendet, die spezielle Zeichen wie zum Beispiel das Anführungszeichen in das entsprechende Entity &quot; umwandelt. Dies ist zum Beispiel auch dann wichtig, wenn Sie ein normales Texteingabefeld mit einem Text vorbelegen wollen.

**Dateiupload**

Für den Dateiupload gilt die Besonderheit, dass er nur mit der POST-Methode funktioniert und das <form>-Tag den Parameter enctype=“multipart/form-data“ enthalten muß. Nach dem Upload wird die Datei temporär auf den Server kopiert und sollte von Ihrem Script an einen anderen Ort kopiert werden, da PHP am Ende des PHP-Scripts diese Datei vom Server löscht. Mit einer hidden Formularvariable MAX\_FILE\_SIZE können Sie den Browser anweisen, nur Dateien bis zu dieser Größe (in Bytes) upzuloaden. Bitte beachten Sie, dass ein Browser sich nicht unbedingt daran halten muß. Auf der Serverseite können Sie in der php.ini die Maximalgröße von uploadenden Files festlegen.

<form enctype=“multipart/form-data“ method=“POST“>

<input type=”hidden” name=”MAX\_FILE\_SIZE” value=”4096”>

Datei: <input type=”file” name=”userfile”>

<input type=”submit” value=”Upload”>

</form>

Nach erfolgtem Submit erhalten Sie in PHP folgende Variablen:

• $userfile: die **temporäre** Datei auf dem Server

• $userfile\_name: der **Originalname** der Datei

• $userfile\_size: die Größe der Datei in Bytes

• $userfile\_type: der MIME-Typ der Datei (z.B. „image/gif“)

Ist register\_globals (ein Schalter, um externe Variablen in den globalen Namespace zu importieren und z.B. automatisch als Variable $userfile verfügbar zu machen) auf off, so gibt es im Array $HTTP\_POST\_FILES die entsprechenden Daten:

• $HTTP\_POST\_FILES[„userfile”][“name”]: der Originalname

• $HTTP\_POST\_FILES[“userfile”][“tmp\_name”]: temporärer Name auf dem Server

• $HTTP\_POST\_FILES[“userfile”][“size”]: Größe der Datei in Bytes

• $HTTP\_POST\_FILES[„userfile“][„type“]: MIME-Typ der Datei

In der Regel werden die hochgeladenen Dateien in das temporäre Verzeichnis des Betriebssystems (z.B. /tmp/) kopiert. Dies kann jedoch über die Direktive upload\_tmp\_dir in der php.ini geändert werden.

Um zu verifizieren, dass die Datei auch wirklich eine Datei ist, die uploaded wurde, existiert die Funktion is\_uploaded\_file(string filename). Sie liefert TRUE zurück, wenn die Datei per HTTP POST hochgeladen wurde.

Mit move\_uploaded\_file(string filename, string destination) können Sie die hochgeladene Datei an einen anderen Ort kopieren. move\_uploaded\_file() testet dabei direkt via is\_uploaded\_file(), ob die Datei tatsächlich von einem HTTP POST Request stammt.

**Image als Absendeknopf**

Normale HTML Submit Buttons sind oftmals grau und grässlich. Stattdessen möchte man oftmals lieber eine eigene Grafik als Absendeknopf einbauen. Dazu gibt es den input type „image“:

<input type=“image“ name=”bild” src=“/img/return.gif“ border=“0“>

Das border=”0” sollte man niemals vergessen, wenn man keinen Rand um das Bild haben möchte. Fehlt dieser Parameter, neigen einige ältere Browser dazu, dennoch einen Rand zu zeichnen.

Beim Submit übermittelt der Browser die Variablen name.x und name.y (in obigem Beispiel also bild.x und bild.y).

Da in PHP jedoch keine Variablen mit einem Punkt im Variablennamen existieren können (da der Punkt ein String Operator ist), werden sie von PHP in Unterstriche gewandelt. In obigem Beispiel sind nach dem Submit also die PHP-Variablen $bild\_x und $bild\_y zur Verfügung, *nicht* jedoch $bild. Ein Affenformular sähe also folgendermaßen aus:

<?php

if (isset($submit\_x)) {

print “Submit-Bild wurde gedrückt!”;

} else {

?>

<!— hier GET, um die Übermittlung zu verdeutlichen -->

<form action=“<?php print $PHP\_SELF; ?>” method=”GET”>

<input type=”text” name=”foo”><br>

<input type=”image” name=”submit” src=”/img/return.gif” border=”0”>

</form>

<?php

}

?>

**Typische Anfängerfehler**

In diesem Kapitel finden Sie einige typische Anfängerfehler und Hinweise, wie Sie diese sofort erkennen können.

**Parse error .... in line 10**

Der PHP-Interpreter konnte die erwähnte Zeile nicht parsen, d.h. nicht analysieren/syntaktisch korrekt zerlegen. Oft findet sich dieser Parse error, wenn der Programmierer in dieser Zeile etwas unsinniges stehen hat.

Typisch für diesen parse error sind vergessene einfache oder doppelte Anführungszeichen bei print/printf/sprintf-Anweisungen:

<?php

print “Hallo ich bin kein String;

?>

Sollte sich in der bemängelten Zeile also ein print/echo-Anweisung befinden, so sollten Sie überprüfen, ob auch wirklich alle Anführungszeichen übereinstimmen. Komplizierter wird es, wenn Sie die Zeichen ’ und “ gemischt verwenden:

<?php

print ”<a href=’.$PHP\_SELF.“>Link</a>”;

?>

Hier wurde vor der String Concatenation (also dem Verbinden mehrerer Strings zu einem mittels des Operators **.**) vergessen, den String ordentlich zu beenden. Erschwerend kommt hinzu, dass das einfache Anführungszeichen **’** Teil der auszugebenden HTML-Zeichenkette ist, so dass man beim schnellen Überfliegen des Codes unter Umständen denken könnte „Ist doch alles in Ordnung?“. Korrekt würde das Beispiel also folgender maßen lauten:

<?php

print “<a href=’“.$PHP\_SELF.“>Link</a>;

?>

Oftmals scheint aber die bemängelte Zeile auch nach mehrfachem Hinsehen völlig korrekt zu sein.

<?php

print “Hallo, ich bin ein String; highlight\_source($einedatei);

print “blafasel”; // Zeile 5

?>

(parse error in line 5)

Hier wird also Zeile 5 bemängelt, ob diese völlig korrekt ist. Der Fehler liegt allerdings bereits in Zeile 2: dort wird der angefangene String nicht korrekt beendet und PHP denkt, der String würde in den folgenden Zeilen weiter gehen.

**Tipp:** sollten Ihnen solche parse errors begegnen, so sollten Sie auch immer einige der vorhergehenden Zeilen kontrollieren, ob diese korrekt sind.

**Typische Anfängerfehler II**

Ein weiterer, typischer Anfängerfehler ist es, Resultate von Befehlen nicht zu prüfen. Dies ist zum Beispiel dann notwendig, wenn sie mit fopen() eine Datei öffnen (konnte sie überhaupt geöffnet werden?) oder mit mysql\_query() eine SQL-Query ausführen.

<?php

/\*\*

\* ohne Überprüfung ist es falsch!

\*/

$fp = fopen(„/usr/local/blafasel.txt“);

/\*\*

\* besser ware:

\*/

if (!$fp = fopen(„/usr/local/blafasel.txt“) {

print “Fehler! Konnte die Datei nicht öffnen.<br>“;

} else {

while (!feof($fp)) {

$zeile = fgets($fp, 4096);

print „$zeile<br>“;

}

fclose($fp);

}

?>

Beim Operieren mit Datenbanken ist bei Anfängern oft zu beobachten, dass die Befehle mysql\_connect(), mysql\_query() etc. ohne Überprüfung ausgeführt werden:

<?php

mysql\_connect(„localhost“, „root“, „“);

$res = mysql\_query(“select \* from foo”);

while ($row = mysql\_fetch\_array($res)) {

/\* irgendwas machen … \*/

}

?>

Besser wäre:

<?php

if (!mysql\_connect(„localhost“,“root“,““)) {

/\* Fehlermeldung ausgeben, evtl. Applikation beenden \*/

}

if (!$res = mysql\_query(„select \* from foo“)) {

/\* mysql\_error() liefert direkt eine Fehlermeldung von MySQL \*/

print “MySQL-Fehler: “.mysql\_error();

} else {

while ($row = mysql\_fetch\_array($res)) {

/\* Daten ausgeben. \*/

}

}

?>

Warum diese Überprüfungen sein müssen? Es kommt zum Beispiel oft vor, dass der Query-String nicht SQL-konform ist und MySQL einen parse error liefert.