



Herbert Braun

Schnellkurs PHP

PHP ist überall. Angeblich sollen drei Viertel aller Websites diese Skriptsprache einsetzen, die Anzahl der damit geschriebenen Content-Management-Systeme und Webmaster-Werkzeuge ist Legion. Die folgenden Seiten steigen bei den Grundlagen ein und führen Sie durch ein erstes praxisnahes Mini-Projekt.

Die Anfänge von PHP waren extrem bescheiden: Rasmus Lerdorf suchte Mitte der 90er-Jahre nur nach ein paar simplen Werkzeugen für seinen eigenen Webauftritt. Doch spätestens mit Version 3, bei dem Andi Gutmann und Zeev Suraski mit an Bord waren, setzte PHP zur Weltherrschaft an, und PHP4 aus dem Jahr 2000 erreichte eine enorme Verbreitung – was für die Nachfolgeversion zum Problem wurde, da sie nicht zu 100 Prozent abwärtskompatibel war und die Anbieter von Shared-Hosting-Webspace lange die Umstellung hinauszögerten. Letztlich setzte sich PHP 5, aktuell in Version 5.3.6, aber doch durch. Nur mit den Plänen für PHP 6 geht es nicht so recht weiter.

Mehrere Gründe machen den Erfolg von PHP aus: Zum einen ist es quelloffen und kostenlos, also problemlos verfügbar. Anders als etwa Java oder ASP.NET gibt es sich mit den billigsten Shared-Hosting-Umgebungen zufrieden. Im Unterschied zu Perl, von dem es vieles übernommen hat, ist es spezialisiert auf Website-Programmierung und lässt sich ohne Schwierigkeit in den HTML-Code integrieren. Schließlich das Killer-Argument: PHP galt von Anfang an als einfach. Es versprach, dass damit jeder zum Programmierer werden konnte.

Seine Attraktivität für wenig erfahrene Entwickler führte zu einer Menge schlecht geschriebenem und unsicherem PHP-Code, was dem Ansehen der Sprache zeitweise geschadet hat. Inzwischen sind jedoch ganze Enterprise-Content-Management-Systeme und komplexe Websites mit gewaltigen Nutzerzahlen wie Facebook im Einsatz, die auf PHP aufbauen.

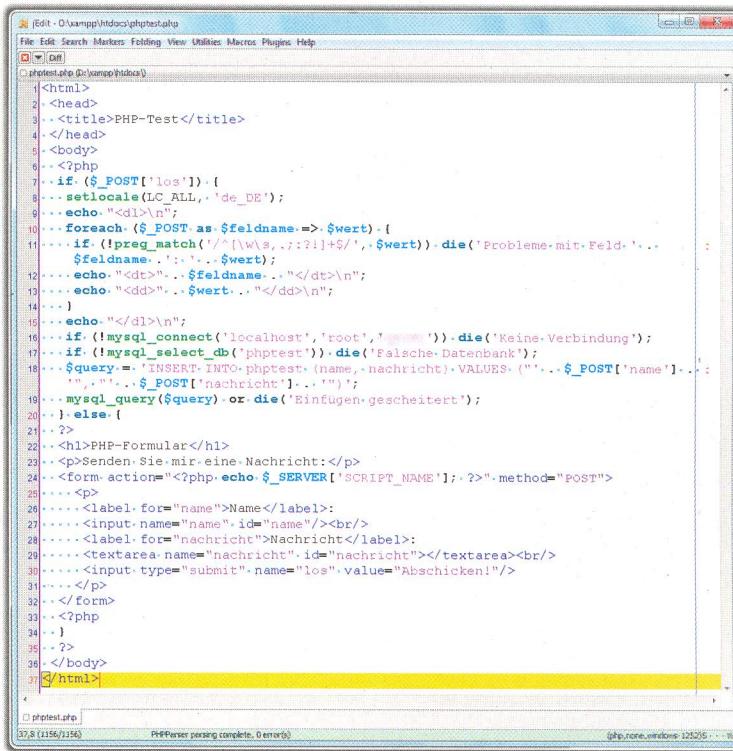
Im Server zu Hause

PHP ist in der Webseite zu Hause. Sein „Hallo, Welt“ ist in HTML eingebettet:

```
<html>
<head>
<title>PHP-Test</title>
</head>
<body>
<p>
Funktioniert PHP?
<?php echo "Ja!"; ?>
</p>
</body>
</html>
```

Der PHP-Code echo "Ja!" ist eingeschlossen in das Tag <?php ... ?>. Dieses fordert den Server auf, er möge den dazwischenstehenden Code bitteschön kompilieren. echo gibt Text aus, sodass das gesamte PHP-Tag am Ende durch Ja! ersetzt wird.

Das funktioniert aber nur, wenn Sie den Code auf einen PHP-fähigen Webserver transportieren – zum Beispiel per FTP auf



Ihren Webspace. Selbst Billig-Webspace ist heutzutage in der Lage, PHP-Code auszuführen. Gerade zu Test- und Entwicklungszwecken empfiehlt sich jedoch die Einrichtung einer lokalen Serverumgebung. Das klingt vielleicht kompliziert, lässt sich aber mit Hilfe der Fertiglösung XAMPP (auf der Heft-DVD) innerhalb von Minuten einrichten und starten. Auch Microsofts WebMatrix (siehe Seite 128) stellt eine lokale Serverumgebung bereit.

Damit der Webserver den PHP-Code ausführt, müssen Sie der Webseite die Dateiendung .php geben (zum Beispiel phptest.php) und sie im Verzeichnis xampp/htdocs oder unterhalb davon ablegen. Gelegentlich sieht man auch Dateiendungen wie .php3, .php4 oder .php5, welche die jeweilige PHP-Version erzwingen. Das ist vor allem interessant, wenn man PHP 5 verwenden will, obwohl der Server per Default das veraltete PHP 4 erwartet – was bei manchen Shared-Hosting-Angeboten immer noch der Fall ist. In XAMPP sollten alle diese Varianten funktionieren. Rufen Sie nun im Browser die Seite localhost/phptest.php auf – wenn XAMPP korrekt läuft, sollte „Funktioniert PHP? Ja!“ im Browser-Fenster erscheinen.

Auch wenn die Webseite die natürliche Lebensumgebung von PHP ist, können Sie es per Kommandozeile nutzen: Steuern Sie mit der Eingabeaufforderung das Verzeichnis xampp\php an und geben Sie php -r "echo 'hallo!'" ein. Selbst die Ausführung von fertigen Skripten ist möglich, zum Beispiel php -f ..\htdocs\phptest.php – das schreibt den HTML-Code einfach in die Konsole.

Ein Editor mit Syntax-Highlighting (hier: jEdit) erleichtert die Arbeit des Programmierens mit PHP ungemein.

Bei den Programmierbeispielen denken Sie sich bitte die Einbettung in die HTML-Seite inklusive der <?php ... ?>-Tags dazu. Sie dürfen auch mehrere Code-Fragmente in die Seite einbauen oder ganz auf das HTML-Gerüst verzichten – schlimmstenfalls hält es der Browser nicht mehr für HTML, sondern für Klartext, was zu PHP-Testzwecken kein Problem ist.

Maximale Ausgabe mit minimalem Code-Aufwand erhalten Sie, wenn Sie einfach phpinfo(); in das PHP-Tag schreiben. Diese Funktion zeigt Ihnen alle möglichen Versionsinformationen, in PHP enthaltene Module und Server-Umgebungsvariablen.

Variabel

Bei den Variablen hat sich PHP von Perl das \$-Zeichen geborgt:

```
$string = 'abc';
$zahl = 123;
$kommazahl = 1.5;
$boole = true;
$array = array(1, 2, 'drei');
$assoz_array = array('eins' => 'one', 'zwei' => 'deux');
```

Anders als bei Perl steht das Dollar-Zeichen vor allen Variablenarten. Die wichtigsten sind Strings, Integer-Zahlen, Gleitkomma-Zahlen, boolesche Werte (true oder false) und Arrays; Letztere definieren Sie mit der Funktion array().

Arrays können die Form einer Liste oder eines assoziativen Arrays annehmen, wo jeder Wert einem einzigartigen Schlüssel zugeordnet ist. Die Schlüssel sind dabei eine

The screenshot shows a detailed explanation of variable scope in PHP. It includes code snippets and text explaining how variables are available in included files and how global variables are referenced from functions.

Auf dieser Seite werden Sie sich bald zu Hause fühlen, wenn Sie der PHP-Virus angesteckt hat: php.net bietet die maßgebliche Dokumentation der Sprache.

optionale Zusatzinformation – um auf den Wert „one“ im Beispiel oben zuzugreifen, könnten Sie `$assoz_array['eins']` schreiben, aber ebenso `$assoz_array[0]`. Eine Besonderheit von PHP ist, dass man Variablen nicht vorher deklarieren kann: Sie entstehen grundsätzlich bei der ersten Verwendung.

PHP integriert sich sehr geschmeidig in den Webauftritt. Daher können Sie mit einigen vordefinierten Variablen mühelos auf Daten der Server-Umgebung zugreifen. Das assoziative Array `$_SERVER` enthält zum Beispiel den Namen der aktuellen Skriptdatei (`$_SERVER["SCRIPT_NAME"]`), den Namen des anfragenden Browsers (`$_SERVER["HTTP_USER_AGENT"]`) oder Details zum Server (`$_SERVER["SERVER_SOFTWARE"]`).

Um die in dieser Variable enthaltenen Schätzungen einzusehen, hilft Ihnen `echo $_SERVER;` nicht weiter: PHP beantwortet diese Frage mit einem lakonischen „Array“. Besser macht es die ungemein praktische Funktion `var_dump()`, der Sie als Funktionsargument das Array übergeben, also:

```
var_dump($_SERVER);
```

Die Ausgabe sieht nur auf den ersten Blick chaotisch aus: In der Quelltextansicht des Browsers sind die mannigfaltigen Inhalte der Variable sauber durch Zeilen getrennt und eingerückt. Ähnlich nützliche Variablen wie `$_SERVER` sind `$_GET` und `$_POST`; sie enthalten Formularinhalte, die mit der jeweiligen HTTP-Methode übermittelt wurden. Wenn Sie beispielsweise an Ihre Skript-URL `?eins=`

`one&zwei=deux` hängen, enthält `$_GET["zwei"]` den String `deux`.

Variablen wie `$_SERVER`, `$_GET` und `$_POST` sind überall zugänglich; sie heißen in PHP „super-global“. Normale Variablen, die Sie selbst definieren, genießen dieses Privileg nicht; ihr Geltungsbereich ist auf den aktuellen Block beschränkt. Das folgende Beispiel funktioniert also nicht wie gewünscht:

```
$string = 'ct';
stringausgabe();
function stringausgabe() {
    echo $string;
}
```

function `Funktionsname()` ... deklariert eine Funktion. Diese sollte eigentlich „ct“ auf den Bildschirm schreiben, aber dieser Codeblock gibt überhaupt nichts aus: `stringausgabe()` hat noch nie etwas von `$string` gehört und erweckt diese Variable automatisch zum Leben, allerdings ohne jeden Wert. Um auf das originale `$string` zuzugreifen, müssen Sie diesen Wunsch mit `global` deutlich machen:

```
$string = 'ct';
stringausgabe();
function stringausgabe() {
    global $string;
    echo $string;
}
```

Die andere Möglichkeit, Variablen in eine Funktion zu übertragen, sind Funktionsargumente:

```
$string = 'ct';
stringausgabe($string);
function stringausgabe($meinstring) {
    echo $meinstring;
}
```

Die zweite Zeile ruft die Funktion `stringausgabe()` mit einem Funktionsargument auf, in diesem Fall eine String-Variablen. Bei der Deklaration der Funktion legen Sie einen Variablenamen fest, unter dem Sie innerhalb der Funktion auf das Argument zugreifen – in diesem Fall `$meinstring`. Mehrere Argumente trennen Sie durch Kommata.

Operatoren

Den Zuweisungsoperator = haben Sie bereits kennengelernt. PHP kennt die üblichen mathematischen Operatoren, insbesondere + - * und /. Auch die folgende Kurzschreibweise hat PHP von anderen Programmiersprachen übernommen:

```
$zahl = 5;
$zahl++;
```

`$zahl++` steht für `$zahl += 1`, was wiederum `$zahl = $zahl + 1` abkürzt. Um zwei Strings zu verbinden, nutzen Sie den Punkt:

```
$welt = 'Welt';
echo 'Hallo ' . $welt;
```

Vergleichsoperatoren wie == (gleich) oder != (ungleich) brauchen Sie vor allem in Kontrollstrukturen:

```
if ($zahl == 1) {
...
} else if ($zahl == 'zwei') {
...
} else {
...
}
```

Bei einzeiligen Blöcken können Sie die geschweiften Klammern auch weglassen:

```
if ($var) echo $wert;
```

if (\$var) ist die Kurzform von if (\$var == true); das trifft zu, wenn \$var irgendeinen Wert zugewiesen bekommen hat, der nicht false, null, der Leer-String oder die Zahl 0 ist; auch der String "0" ist false.

Das bedeutet, dass PHP auch Werte unterschiedlicher Typen vergleichen kann. "2" == 2 funktioniert also. Die Sprache sieht aber die Möglichkeit vor, diese Typwandlung zu verbieten:

```
if ($var === 2) {...}
```

Vergleicht man mit dem dreifachen Gleichheitszeichen, ist "2" nicht mehr 2. Die negative Form dieses Operators heißt !=.

Bei den Schleifen verhält sich PHP wenig überraschend:

```
$i = 0;
while ($i < 5) {
// tu irgendwas
$i++;
}
for ($i = 0; $i < 5; $i++) {
// tu irgendwas
}
```

Die erste Schleife wird ausgeführt, solange \$i kleiner als fünf ist, also genau fünfmal. Die Zeile \$i = 0 ist nicht notwendig, PHP belegt die Variable beim Erzeugen automatisch mit 0. Die for-Schleife funktioniert genauso. Der doppelte Schrägstrich kennzeichnet übrigens einen einzeiligen Kommentar; größere Blöcke können Sie mit /* ... */ auskommieren.

Komfortabel sind foreach-Schleifen:

```
foreach ($mein_array as $wert) {
// tu was mit $wert
}
```

Die Schleife durchläuft jedes Element von \$mein_array und schreibt den jeweiligen Wert in \$wert. Das klappt auch mit assoziativen Arrays:

```
foreach ($mein_assoziatives as $key => $wert) {
echo "$wert steht in $key\n";
}
```

Die Schlüssel-Wert-Paare des jeweiligen Array-Eintrags stehen in \$key und \$wert. Das \n am Ende des Ausgabe-Strings sorgt für einen Zeilenwechsel.

Funktionen

Alle PHP-Funktionen auch nur aufzulisten würde den Rahmen dieses Artikels bereits sprengen. Bevor es an ein kleines Programmierbeispiel geht, sollen wenigstens noch ein paar davon vorgestellt werden.

Häufig braucht man zum Beispiel strpos(), das Strings vergleicht:

```
strpos('Hallo', 'l'); // Wert 2
strpos ('Hallo Welt', 'Hallo'); // Wert 0
strpos ('Hallo', 'h'); // Wert false
```

strpos() sucht im ersten String-Argument nach dem zweiten und gibt dessen Position im ersten zurück, beginnend bei 0; im Falle des Scheiterns antwortet die Funktion mit false. Eine beliebte Falle ist if (strpos("Hallo", "H")) ... – Da die Funktion das Gesuchte an erster Stelle findet, gibt sie die Zahl 0 zurück, die das if als „falsch“ versteht. Korrekt wäre if (strpos("Hallo", "H") != false).

Zum Ersetzen eignet sich str_replace():

```
$ersetzt = str_replace("suche", "ersetze", "Ich suche");
```

Der „Heuhaufen“, in dem gesucht wird, steht merkwürdigerweise an dritter Stelle. Zuvor kommt der Suchstring, dann die Er-

The screenshot illustrates the execution of a PHP script. On the left, a browser window titled 'PHP-Test' shows a simple form with fields for 'Name' (Philipp), 'Nachricht' (Das wahre Glück ist Philosophie.), and a 'Abschicken!' button. On the right, another browser window titled 'PHP-Test' displays the submitted data: 'name' (Philipp), 'nachricht' (Das wahre Glück ist Philosophie.), and 'los' (Abschicken!). Below these windows is a screenshot of the 'phpMyAdmin' interface. It shows a database named 'phptest' with a single table 'phptest'. The table has three columns: 'id', 'name', and 'nachricht'. There is one row with id 6, name 'Philipp', and nachricht 'Das wahre Glück ist Philosophie.'. The SQL query used is: 'SELECT * FROM phptest LIMIT 0, 30'.

Das schmucklose HTML-Formular verwandelt sich nach dem Abschicken in eine Liste der übergebenen Inhalte. Ein Blick in PHPMyAdmin beweist, dass diese auch in der Datenbank angekommen sind.

setzung. Das Beispiel schreibt in \$ersetzt „Ich ersetze“.

Besonders komfortabel gelingt in PHP das Verschicken von E-Mails:

```
mail('hotline@ct.de', 'Riesenproblem', $langertext);
```

Diese Funktion schickt an die im ersten Argument genannte Adresse eine Mail, bei der das zweite und dritte Argument Betreff und Mailtext enthalten. Zusätzliche Mail-Header können im vierten Argument stehen, zum Beispiel "From: webmaster@ct.de". Hat alles geklappt, gibt mail() ein true zurück. In XAMPP wird das erst funktionieren, wenn Sie in der php.ini einen SMTP-Server eintragen.

PHP-Poststelle

Tiefer ins Detail geht es mit einem typischen PHP-Einsatzszenario: einem Kontaktformular. Es soll zwei Felder für Name und Nachricht enthalten. Als Rückmeldung möge es dem Benutzer seine Eingaben zeigen, während diese im Hintergrund in der Datenbank gespeichert werden – und all das möglichst ohne jedem hergelaufenen Hobby-Hacker die Tür sperrangelweit aufzumachen.

Das Herzstück des HTML-Codes sieht ungefähr so aus:

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>" method="POST">
```

```
Name: <input name="name"/><br/>
Nachricht: <textarea name="nachricht"></textarea><br/>
<input type="submit" name="los" value="Abschicken!" />
</form>
```

Beim Abschicken gehen die Inhalte von drei Eingabefeldern durch die Leitung – name, nachricht und der Submit-Button los, dessen Inhalt nicht weiter interessiert. Das Formular wird mit der HTTP-Methode POST versendet; die in \$_SERVER['SCRIPT_NAME'] gespeicherte Zieladresse ist das Skript selbst. Das heißt: Wenn irgendwelche POST-Daten im Skript ankommen, soll es sich völlig anders verhalten als wenn nicht. Die Grundstruktur muss also etwa so aussehen:

```
<body>
<?php
if ($_POST['los']) {
// Formulardaten verarbeiten ...
} else {
?>
HTML-Formular ...
<?php
}
?>
</body>
```

Ein if ... else-Konstrukt umspannt den ganzen HTML-Body. Als Wasserscheide dient die Frage, ob per POST ein Formularfeld namens los angekommen ist – falls ja, müssen die For-

mulardaten verarbeitet werden, ansonsten erscheint das HTML-Formular. Sie dürfen PHP-Blöcke durch beliebig viel HTML-Code unterbrechen, aber vergessen Sie nicht, offene Klammern wieder zu schließen.

Den Inhalt des Formulars geben Sie mit einer einfachen foreach-Schleife aus:

```
if ($_POST['los']) {
    echo "<dl>\n";
    foreach ($_POST as $fieldname => $wert) {
        echo "<dt>" . $fieldname . "</dt>\n";
        echo "<dd>" . $wert . "</dd>\n";
    }
    echo "</dl>\n";
} else {
```

Aus jedem Eintrag in \$_POST holt die Schleife \$fieldname und \$wert heraus. Die Ausgabe nutzt eine HTML-Definitionsliste (<dl>).

Bevor Sie diese Daten in die Datenbank schreiben, möchten Sie vielleicht noch kontrollieren, dass dort kein gefährlicher Unfug landet. Wenn Sie schon in der foreach-Schleife jedes Element einzeln anfassen, können Sie es auch gleich unter die Lupe nehmen. Das beste Werkzeug dafür sind reguläre Ausdrücke, die in den meisten Programmiersprachen zur Verfügung stehen. PHP kennt Perl-kompatible reguläre Ausdrücke; sie sind in einer Reihe von Funktionen implementiert, die alle mit preg_ beginnen:

```
foreach (...) {
    if (!preg_match('/^[\w\$\.,;?!]+$/i', $wert))
        die("Probleme mit Feld $fieldname: $wert");
// ...
}
```

preg_match() enthält als erstes Argument den regulären Ausdruck, als zweites den zu untersuchenden String. In diesem Fall darf dieser nicht leer sein und muss aus Buchstaben, Ziffern, Unterstrichen, Leerzeichen und einigen Satzzeichen bestehen. Eine ausführliche Erläuterung der – äußerst nützlichen – regulären Ausdrücke finden Sie zum Beispiel in der offiziellen PHP-Dokumentation (siehe Link am Ende des Artikels). Scheitert der Mustervergleich im Beispiel, weil ein verbotenes (oder gar kein) Zeichen in \$wert steht, bricht das Skript sofort mit einer Fehlermeldung ab.

Datenbank

Nun sollen die Benutzereingaben in der Datenbank festgehalten werden. Bevor Sie sich die MySQL-Schnittstelle von PHP ansehen, sollten Sie aber erst einmal eine Datenbank anlegen. In XAMPP und in den meisten Hosting-Paketen gibt es für MySQL eine grafische Weboberfläche namens PHPMyAdmin, die Sie unter <http://localhost/phpmyadmin> erreichen können sollten. Wie der Name schon sagt, greift PHPMyAdmin ebenfalls

mit PHP-Werkzeugen auf die Datenbank zu. WebMatrix hat einen eigenen Datenbank-Verwalter, auf den wir hier nicht näher eingehen.

Auf der Startseite von PHPMyAdmin können Sie eine neue Datenbank anlegen, die zum Beispiel „phptest“ heißen kann. Die Datenbank benötigt mindestens eine Tabelle, die der Einfachheit halber den gleichen Namen bekommt. Sie braucht drei Felder: einen Index namens id und die beiden zu speichernden Felder name und nachricht. id ist ein Zahlenfeld (INT), dessen Werte per auto increment (unter „Extra“) automatisch eingetragen werden. Bei den Radio-Buttons am Ende wählen Sie das Schlüsselsymbol für „Primärschlüssel“. name wird ein VARCHAR-Feld mit der Länge 255, nachricht ein TEXT-Feld, das noch mehr Platz bietet. Das sollte für einfache Datenbankspielereien genügen.

Zurück zum PHP-Code. Nachdem der Inhalt der Felder eingelesen, geprüft und ausgegeben wurde, stellen Sie hiermit eine Verbindung zur Datenbank her:

```
if (!mysql_connect('localhost','user','password'))
    die('Keine Verbindung');
```

Die Funktion mysql_connect() stellt die Verbindung zu MySQL her. Dazu benötigt sie den Datenbank-Host (localhost funktioniert, wenn die Datenbank auf der gleichen Maschine wie der Webserver läuft), den Benutzernamen und das Passwort. Beim lokalen Herumspielen müssen Sie nicht groß auf Sicherheit achten, im Web dagegen sehr. Gelingt es mysql_connect(), eine Verbindung herzustellen, gibt es true zurück. Der vorangestellte Operator ! verkehrt diesen Wert ins Gegenteil: Ist die Verbindung gescheitert, trifft die Bedingung zu und PHP führt die die()-Funktion aus. Diese bricht das Skript sofort mit dem angegebenen Text ab.

Nun haben Sie Verbindung zum Datenbankserver, aber noch nicht zu einer Datenbank. Das geht so:

```
if (!mysql_select_db('phptest')) die('Falsche Datenbank');
```

Der Funktionsname mysql_select_db() ist ja fast selbsterklärend. An neuralgischen Punkten wie diesem ist es sinnvoll zu kontrollieren, ob die Anweisung erfolgreich war. Diese Datenbank befüllen Sie mit einer SQL-Anweisung, die etwa so aussieht:

```
INSERT INTO phptest (name, nachricht) VALUES ("Anja",
                                              "Hallo, Welt")
```

In PHP verpacken Sie diese Abfrage als String. Beide POST-Felder name und nachricht enthalten Texte und müssen in SQL in Anführungszeichen stehen, was ein bisschen mühsam und fehlerträchtig ist:

```
$query = 'INSERT INTO phptest (name, nachricht) VALUES ("'
        . $_POST['name'] . '", "' . $_POST['nachricht'] . ")';
```

An dieser Stelle können Sie mit die(\$query) prüfen, ob alles richtig aussieht. Falls ja, müssen Sie den Abfrage-String nur noch an die Datenbank weiterreichen:

```
if (!mysql_query($query)) die('Einfügen gescheitert');
```

Auch mysql_query() gibt true oder false zurück. Natürlich können Sie mit dem mysql-Modul die Datenbank auch auslesen. Wie das geht, erfahren Sie auf www.php.net, der offiziellen Dokumentation.

Der Elefant im Zimmer

PHP ist das Windows unter den Programmiersprachen: Obwohl viele Experten von Anfang an die Nase darüber rümpften, konnten sie seinen phänomenalen Erfolg nicht stoppen. Während es nach wie vor bei Hobby- und Gelegenheitsprogrammierern verankert ist, dringt es von dort aus in immer komplexere Projekte ein. Aus Unternehmenssicht ist ein großer Vorteil von PHP, dass es viele vergleichsweise billige Programmierer dafür gibt. Entwickler profitieren dagegen von der riesigen Community und den zahllosen Fertiglösungen und Frameworks, die das Arbeiten erleichtern.

Hemmungslos haben die PHP-Macher unterschiedliche Konzepte zusammengeführt. So mögen die Puristen angesichts der PHP aufgepropften Objektorientierung die Hände vors Gesicht schlagen: Bei vielen Entwicklern kommt dieser Eklektizismus offenbar gut an. Unglaubliche 5800 Funktionen enthält die Standarddistribution der Sprache. Weder diese Zahl noch die uneinheitliche Namensgebung und Anordnung der Argumente (man vergleiche etwa strpos(), str_replace() und parse_str()) konnten den Ruf von PHP erschüttern, eine „einfache“ Sprache zu sein.

Fortschritte bei der Weiterentwicklung der Sprache gibt es durchaus. So sind problematische Altlasten wie die PHP-Einstellungen magic_quotes und register_globals – vor allem Letztere ist ein Sicherheitsrisiko – längst standardmäßig deaktiviert. Und als Ersatz für die alte MySQL-Bibliothek, die für SQL-Injektionen anfällig ist, gibt es das in PHP 5 eingeführte MySQLi-Modul, auch wenn dieses bislang eher stiefmütterlich dokumentiert ist.

PHP hat bereits etliche „PHP-Killer“ (man denke etwa an Ruby on Rails) unbeschadet überlebt. Es sieht so aus, als würde auch in ein paar Jahren noch viel in dieser Sprache programmiert werden. Wer also Anwendungen fürs Web schreiben will und sich nicht mit ASP.NET oder Java anfreunden kann, dürfte kaum an PHP vorbeikommen – vor allem, wenn mehrere Entwickler an dem Projekt arbeiten sollen. (heb)

www.ct.de/cs1109104