PEARSON Education

Markt+Technik

magnum

PHP 5 & MySQL 4.1

CHRISTINE PEYTON ANDRE MÖLLER

Markt+Technik

kompakt
komplett
kompetent

3 PHP – Die Grundlagen

Eine neue Programmiersprache zu lernen, heißt, zu versuchen, zunächst die Grundlagen der Sprache zu verstehen, den prinzipiellen Aufbau eines Scripts, elementare Befehle, auch Begriffe, mit denen Sie immer wieder konfrontiert sind etc. Diese Grundlagen sollen in den nächsten Kapiteln geklärt werden. Sie erfahren zunächst, wie ein PHP-Script aufgebaut wird und wie Sie HTML und PHP kombinieren. Dann werden beispielhaft wichtige und gebräuchliche Befehle vorgestellt und an kleinen Beispielen wird kurz erläutert, wie Sie diese Befehle benutzen.

Ein wichtiger Baustein von PHP und anderen Programmiersprachen sind Variablen. Wir klären also, was Variablen sind, welche unterschiedlichen Variablentypen bzw. Datentypen es gibt und wie Sie sie einsetzen und bearbeiten können.

Zu guter Letzt wird in diesem Kapitel von Arrays die Rede sein. Arrays nehmen mehrere Variablen gleichzeitig auf, man kann auf einzelne Elemente zugreifen und sie durchlaufen. Sie werden sehen, dass Arrays äußerst praktische Gebilde sind, auf die beim Programmieren nicht verzichtet werden kann.

Im allerletzten Abschnitt kommen wir zu den regulären Ausdrücken. Leser mit Programmiererfahrung (beispielsweise Perl) wissen, was gemeint ist, als Einsteiger können Sie an dieser Stelle kaum ahnen, worum es sich handelt. Reguläre Ausdrücke gehören eher in die Welt des fortgeschrittenen Programmierens, da sie aber in bestimmten Situationen sehr nützlich sein können, werden wir Sie hier zumindest mit den Grundlagen vertraut machen und gegen Ende im Kapitel 23 *Tipps und Tricks* den Gebrauch an Beispielen demonstrieren.

Eines dieser Beispiele ist – zugegebenermaßen – ausufernder geworden, als ursprünglich beabsichtigt. Wir testen mithilfe eines regulären Ausdrucks die korrekte Eingabe einer E-Mail-Adresse; dabei haben wir versucht, alle erlaubten und unerlaubten Zeichen und die diversen Kombinationsmöglichkeiten abzudecken. Dazu ist – wie sich zeigte – ein Ausdruck erforderlich, der es in sich hat! Wer Spaß hat an ein bisschen Tüftelei, kann versuchen, das Beispiel nachzuvollziehen (am besten mitzuspielen). Als Belohnung winkt ein regulärer Ausdruck, den Sie genauso auf Ihren zukünftigen Webseiten einsetzen könn(t)en, sofern Besucher dort ihre E-Mail-Adresse eingeben. (Ja, wir wissen, es gibt verlockendere Belohnungen, oder!)

Noch ein Hinweis: Dieses Kapitel deckt – wie gesagt – die allerersten Grundlagen ab, behandelt aber bei weitem nicht alle Befehle und sonstigen Möglichkeiten von PHP. Zu anderen elementaren Themen (eingebaute Funktionen, Kontrollstrukturen) kommen wir dann in den nächsten beiden Kapiteln. Im Prinzip folgt das Buch – wie im Vorwort bereits geschildert – einem projektorientierten Konzept. Es versucht, Ihnen PHP an praxisnahen Beispielen unter dem Einsatz von Befehlen, die häufig gebraucht werden, nahe zu bringen. Falls Sie hier dennoch eine Erklä-

rung schmerzlich vermissen: Werfen Sie einen Blick in die Befehlsreferenz, dort finden Sie unter Umständen das Gesuchte.

Ein weiterer Hinweis zu den Abbildungen in diesem Kapitel: Die Screenshots wurden mit einem kleinen »Trick« (sprich: mit einem Frameset) teilweise so gemacht, dass Sie nicht nur die eigentliche Ausgabe sehen, sondern in der rechten Seite des Browers die mit dem Editor geschriebenen Scripts. Teilweise sind die Fenster auch dreigeteilt und unten befindet sich der von PHP erzeugte HTML-Quellcode.

3.1 Die Scripts testen

Während Sie beim Erstellen von HTML-Seiten diese Seiten direkt im Browser betrachten können, benötigen Sie zum Testen von PHP-Dateien einen Server, der PHP unterstützt. Arbeiten Sie an Ihrem heimischen Computer und haben Sie – entgegen dem, was in Kapitel 2 beschrieben wurde – keine Testumgebung installiert, müssen Sie die Scriptdateien also unter Umständen auf den Server Ihres Providers hochladen. Sie benutzen dazu irgendein Standard-FTP-Programm, z.B. WS_FTP. Die Konfiguration von Apache und PHP entfällt dann natürlich und man kann quasi »life« testen. Dies bedeutet auch, dass die Testumgebung genauso konfiguriert ist wie die Umgebung, in der die Scripts nachher ausgeführt werden sollen. Der Nachteil liegt jedoch klar auf der Hand: ohne Flatrate unter Umständen ein teuer Spaß! Arbeiten Sie in einer heimischen Testumgebung, können Sie die PHP-Dateien auf keinen Fall per Doppelklick im Windows Explorer aufrufen – wie Sie es mit HTML-Dateien machen können – sondern Sie müssen sie über die URL (z.B. `http://localhost/ihredatei.php`) aufrufen und damit den Webserver und PHP involvieren.


HINWEIS

FTP (File Transfer Protocol) ist ein Dateiübertragungsprotokoll, mit dem Sie Texte oder binäre Dateien zwischen Ihrem Computer und einem FTP-Computer im Internet austauschen können. In den meisten Fällen benötigen Sie für den FTP-Zugang eine bestimmte Zugangsberechtigung; es gibt jedoch auch so genannte anonyme FTP-Server, die allen Benutzern Zugang gewähren, damit diese beispielsweise kostenlose Software herunterladen oder Dokumente beziehen können.

Sobald Sie eine Verbindung zum FTP-Server aufgebaut haben, können Sie Dateien mehr oder minder so, wie im Windows Explorer üblich, in ein entsprechendes Verzeichnis auf den Server kopieren.

3.2 Das Grundgerüst eines PHP-Scripts

Sie brauchen zum Schreiben eines PHP-Scripts, wie Sie es von HTML gewohnt sind, einen Texteditor – wie Sie wissen, bei Windows in der Regel zu finden im Startmenü unter *Zubehör*. Ein reines PHP-Script ohne HTML-Elemente sieht dann in seiner Struktur zunächst ganz einfach aus. Der ausführbare PHP-Code wird in die folgenden Anfangs- und Endzeichen eingeschlossen:

```
<?php
PHP-Code;
?>
```

Wie Sie noch sehen werden, spielt die Groß- und Kleinschreibung vielfach eine Rolle, allerdings nicht beim Anfangszeichen. Es könnte auch `<?PHP` heißen. Eventuell können Sie auch bei Ihrem Provider nachfragen (bzw. in der Tabelle mit Informationen über die spezifische Installationsumgebung nachsehen, die im nächsten Abschnitt vorgestellt wird), inwieweit Sie kurze Tags benutzen können, also `<?` und `?>` an Stelle von `<?php` und `?>`, oder ob ASP-Tags akzeptiert werden: `<%` und `%>`.

Jede PHP-Anweisung wird mit einem Semikolon beendet. Damit wird PHP quasi mitgeteilt, wo der Befehl, der ausgeführt werden soll, zu Ende ist. Es bietet sich wegen der Übersichtlichkeit an, danach jeweils in eine neue Zeile zu springen, auch wenn dies von der Sache her nicht erforderlich ist. Ein Semikolon nach einem Code zu vergessen, ist ein oft vorkommender Fehler; am besten, es geht Ihnen in »Fleisch und Blut« über, jede Anweisung mit einem Semikolon abzuschließen. Manche Programmierer schreiben das Semikolon grundsätzlich zuerst und dann die anderen Eingaben der Programmzeile. Dies ist ein Trick, der sicherlich dafür sorgt, das Semikolon weniger häufig zu vergessen!

Zusammen mit HTML enthält ein Script die entsprechenden HTML-Tags und den PHP-Code, der in den Body-Container geschrieben und abgegrenzt wird. Das sieht dann als Grundgerüst so aus:

```
<html>
<head>
<title>HTML_PHP</title>
</head>
<body>

<?php

PHP-Code;

?>

</body></html>
```

Sie können beliebig viele PHP-Codes in ein Dokument einfügen und mit den jeweils benötigten HTML-Tags kombinieren. Sie müssen nur jedes Mal auf das Startzeichen `<?php` und das Endzeichen `?>` achten. Oftmals ist auch eine andere Reihenfolge als die hier vorgestellte sinnvoll, z.B. das Script mit `<?php` zu beginnen, anstatt mit dem üblichen HTML-Header. Sie werden weiter hinten in den Kapiteln, in denen konkrete Aufgaben gelöst werden, Beispiele finden (Cookies setzen, Header-Informationen angeben), in denen so vorgegangen wird.

Grundsätzlich arbeitet der PHP-Prozessor die Datei der Reihe nach von oben bis unten ab; wann Sie welche Anweisung wohin schreiben, spielt also eine große Rolle. Reines HTML wird unverändert zurückgegeben, die PHP-Anweisungen werden ausgewertet und ausgeführt.

Sie werden feststellen, dass die Arbeit in einem reinen Texteditor wie z.B. dem Windows-Editor nicht das »Gelbe vom Ei« ist. Es gibt eine Reihe von Freeware-Programmen, die einen erstaunlichen Leistungsumfang bieten und wesentlich komfortabler als reine Text- oder HTML-Editoren sind. Welcher Editor der »richtige« für Sie ist, ist letztendlich Geschmacksache. In der folgenden Tabelle listen wir einige dieser Editoren (Freeware/Shareware) auf.

Bezeichnung	Beschreibung
PHPEdit	recht leistungsstark, trotzdem einfach zu handhaben. Syntax wird hervorgehoben (Syntax-Highlighting), Code-Vervollständigung, Debugger integriert
PHP CODER	stellt zahlreiche Features zur Verfügung, enthält aber auch kleine Fehler
Weaverslave	bietet gute Unterstützungen, neben Syntax-Highlighting eine Unterstützung für Debugger, zahlreiche optionale Einstellungen sowie viele Helferlein für den Umgang mit MySQL, PHP, HTML etc. Praktisch: Man kann nach verschiedenen Quellcodes unterscheiden.
sx Edit	beschränkt sich auf die wesentlichen Funktionen, gut zu bedienen, bietet – wie die meisten Editoren – Syntax-Highlighting

Tabelle 3.1: PHP-Editoren – eine Auswahl

3.3 Speichern

HTML-Seiten, die PHP-Elemente enthalten, speichern Sie mit der Erweiterung *.php*. Dies ist die Standarderweiterung für PHP, und deswegen werden wir in diesem Buch diese Erweiterung benutzen. Sie können die meisten Webserver so konfigurieren, dass sie auch Dateien mit anderen Erweiterungen als PHP-Scripts ansehen.

Sofern Sie nicht direkt am Server arbeiten, muss das Dokument mithilfe eines FTP-Programms auf den Server hochgeladen werden. Davon war weiter oben schon die Rede. Bei den folgenden Beschreibungen gehen wir davon aus, dass Sie am Server arbeiten, sodass Sie die Scripts testen können, indem Sie die Dateien einfach im Browser aufrufen. Sind Sie der Beschreibung der Installation in Kapitel 2 gefolgt, haben Sie als Webserver den Apache auf Ihrem Rechner laufen. Beachten Sie, dass Sie die Datei mithilfe des Webservers aufrufen müssen, damit das enthaltene PHP-Script ausgeführt werden kann. Speichern Sie die Datei also in den Ordner, den Sie bei der Einrichtung von Apache als DocumentRoot festgelegt haben. Geben Sie zum Aufrufen der Datei in der Adressleiste des Browsers z.B. `http://localhost/dateiname.php` ein. Wenn Sie das Script direkt von der Festplatte über z.B. C:\PROGRAMME\APACHE GROUP\APACHE\HTDOCS\DATEINAME.PHP aufrufen, wird das PHP-Script nicht ausgeführt. (Folglich heißt es im Buch nur: Speichern Sie das Script und testen Sie es im Browser oder so ähnlich.)

Sie können in manchen Fällen auch zwei separate Dokumente erstellen, ein HTML-Dokument und ein PHP-Dokument. In dem HTML-Dokument wird dann auf die PHP-Datei, die z.B. die Auswertungen übernimmt, verwiesen. Diese Methode hat allerdings viele Nachteile bei der Verarbeitung. Dennoch: da diese Verfahrensweise das Zusammenspiel von PHP und HTML bzw. die Übergabe von Inhalten (beispielsweise eines Formularfeldes) an die PHP-Scriptdatei sehr anschaulich macht, werden wir sie für die Lösung der ersten konkreten Aufgabe verwenden. In Kapitel 6, *Ein Kontaktformular*, erstellen wir zwei Dokumente, anstatt nur eine einzige PHP-Datei, die den PHP-Programmcode und HTML kombiniert.

3.4 Ausgabe

Damit Sie so bald wie möglich ein Script im Browser testen können, lernen Sie hier – und in den meisten Büchern über PHP – als Erstes den PHP-Befehl `echo` kennen. `echo` gibt alle Zeichenketten und auch Zahlen im Browser aus. Alternativ zu `echo` können Sie `print` schreiben, zwischen diesen beiden Befehlen gibt es keinen wesentlichen Unterschied. (Wir verwenden mal den einen, mal den anderen Befehl – wie gesagt, es spielt in unseren Fällen keine Rolle). Unter einer Zeichenkette, auch als String bezeichnet, versteht man in PHP eine Reihe von Zeichen, die aus einer beliebigen Kombination von Buchstaben, Zahlen, Symbolen und Leerstellen (und Variablen) bestehen kann und in einfache oder doppelte Anführungszeichen eingeschlossen wird.

Schauen Sie sich das einmal an:

1. Öffnen Sie ein leeres Dokument und schreiben Sie:

```
<?php
echo "ich lerne PHP!";
?>
```

2. Speichern Sie das Dokument als Datei mit der Erweiterung *.php*.
3. Öffnen Sie nun diese Datei im Browser. Denken Sie daran, dass Sie die Datei direkt über den Webserver z.B. mit: `http://localhost/dateiname.php` aufrufen. Dort müssten Sie nun lesen:

   ich lerne PHP!

Um eine bestimmte Formatierung (und/oder Seitengestaltung) zu erreichen, können Sie PHP und HTML auch mischen, indem Sie zwischen PHP und HTML hin und her wechseln. Eine Möglichkeit könnte so aussehen:

```
<html>
<head>
<title>HTML_PHP</title>
</head>
<body>
```

```
<b>
<?php
echo "ich lerne PHP";
?>
</b>

<p><b>hier kommt normale HTML-Ausgabe</b></p>

<?php
echo "hier steht ein neuer PHP-Code";
?>

</body></html>
```

Bei dieser Methode beenden Sie also PHP, schreiben den HTML-Teil mit den jeweils benötigten Tags und beginnen PHP erneut. Auch bei viel Text empfiehlt sich ein – unter Umständen auch – beständiger Wechsel zwischen HTML und PHP. Es macht keinen Sinn, Textmengen, die einfach angezeigt werden könnten, unbedingt mit `echo` bzw. `print` durch PHP verarbeiten (parsen) zu lassen.

Es ist aber auch möglich, HTML mit Hilfe von PHP auszugeben. Dies ist für PHP-Einsteiger mitunter etwas verwirrend. Sie müssen sich klar machen, dass in dem Fall alles, was der Browser anzeigen soll, Teil des PHP-Codes sein und ausgegeben werden muss, auch Formatierungen, Zeilenumbrüche und Ähnliches. Ein Scriptfragment würde dann etwa so aussehen:

```
<?php

echo "<b>ich lerne PHP</b>";
echo "<h2>eine Überschrift</h2>";
echo "<hr>";
echo "neuer PHP-Code";

?>
```

Der Browser gibt dann die Seite wie in Bild 3.1 zu sehen aus:

Der Befehl `echo` funktioniert durch die Verwendung des HTML-Tags für einen Zeilenumbruch auch über mehrere Zeilen. Wenn Sie schreiben:

```
echo "mein Hut, der hat vier Ecken, <br> vier Ecken hat mein Hut!";
```

wird als Ausgabe im Browser zu lesen sein:

> Mein Hut, der hat vier Ecken,
> vier Ecken hat mein Hut!

Andererseits können Sie, um den Quelltext zu strukturieren, den Befehl `echo` auch über mehrere Quelltextzeilen verwenden. Wenn Sie Folgendes schreiben, wird alles innerhalb der Anführungszeichen ausgegeben, so, als ob es in einer Zeile stehen würde:

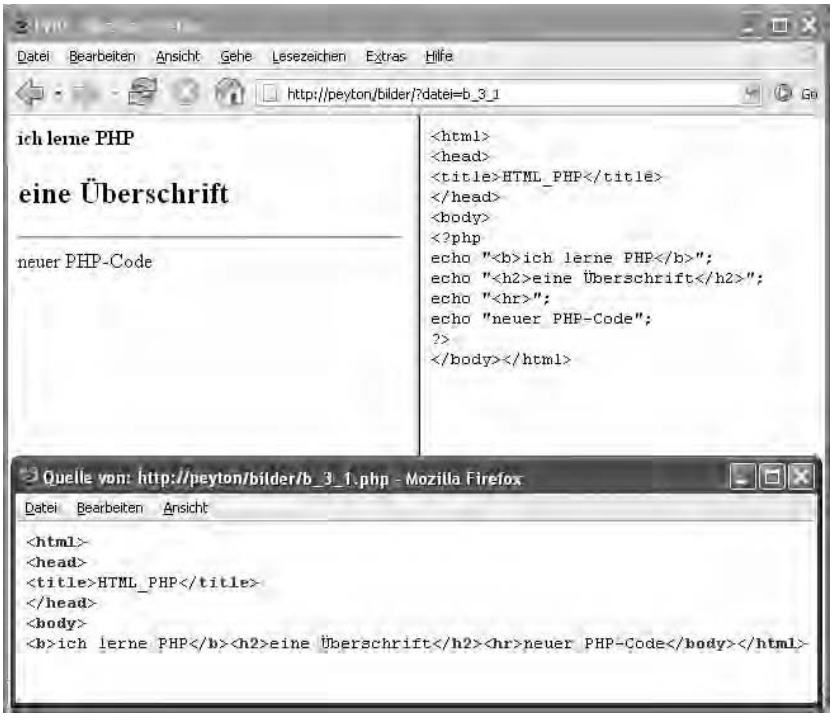

Bild 3.1: Die Ausgabe im Browser – ein fett formatierter Satz (<b>), eine Überschrift und eine horizontale Linie <hr>

```
echo "
mein Hut, der hat vier Ecken,<br>
vier Ecken
 hat mein Hut!
";
```

Im Browser wird als Ausgabe ebenfalls zu lesen sein:

Mein Hut, der hat vier Ecken,
vier Ecken hat mein Hut!

HINWEIS

Leerzeilen im Script haben keinerlei Einfluss auf das Erscheinungsbild der Seite, sie können aber das Script selbst übersichtlicher machen. Auch per Tabulator eingerückte Teile erhöhen die Lesbarkeit eines Scripts und sicherlich werden Sie feststellen, dass es sinnvoll ist, HTML und PHP optisch zu trennen. Aus drucktechnischen Gründen können wir selbst diese Empfehlung hier nicht konsequent einhalten, die meisten abgebildeten Scripts sind optisch also nicht vorbildlich!

3.5 PHP-Info

Es gibt eine gute Möglichkeit bzw. eine spezielle Funktion, sich grundlegende Informationen über PHP einzuholen. Diese Funktion lautet:

```
phpinfo()
```

Mit dieser Funktion wird eine Tabelle an den Browser gesendet, die Informationen über die spezifische PHP-Installation auf dem fraglichen Server enthält. Sie können durch den Einsatz dieser Funktion eine ganze Menge über die Server-Umgebung und die Konfiguration erfahren, z.B. welche Erweiterungen benutzt werden können, welche Datenbank verwendet wird etc. Vor allem, wenn Sie nicht am Server arbeiten und PHP nicht selbst installiert haben, bietet es sich an, einen Blick auf diese Tabelle zu werfen. Dafür benötigen Sie ein ganz einfaches »Script«:

```
<?php

phpinfo();

?>
```

Der Browser zeigt daraufhin die erwähnte Tabelle mit den Informationen über PHP. Riskieren Sie ruhig einen Blick. Abbildung 3.2 gibt einen Ausschnitt der Tabelle wieder.

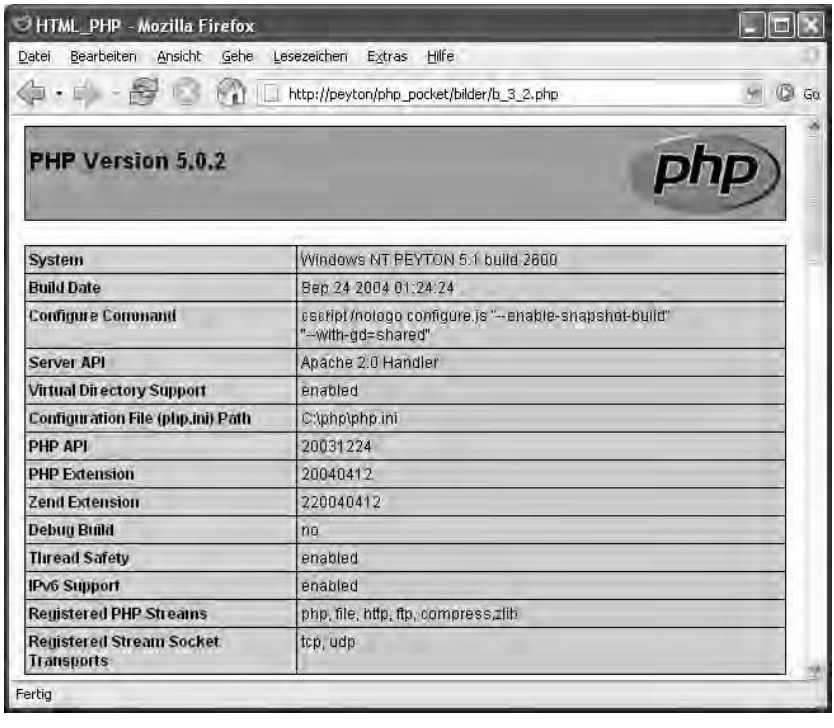

Bild 3.2: Mit phpinfo() können Sie sich Informationen über die spezifische PHP-Installation ausgeben lassen

3.6 Sonderzeichen

Zeichenketten werden – wie Sie gesehen haben – in Anführungszeichen gesetzt. Das bringt offensichtlich Probleme mit sich, wenn ein Textstück tatsächlich in Anführungszeichen ausgegeben werden soll. Als Lösung können Sie zwei Methoden anwenden. Entweder Sie verwenden doppelte und einfache Anführungszeichen, beispielsweise doppelte Anführungszeichen für die Zeichenkette und einfache für das Textstück (oder vice versa):

```
"ich lerne 'PHP'"; oder 'ich lerne "PHP"';
```

Oder Sie verwenden als Maskierungszeichen den Backslash. Dieser wird vor die Anführungszeichen für das Textstück gesetzt. Damit sagen Sie PHP, dass die Anführungszeichen ausgegeben, aber nicht als Beginn oder Ende eines Strings (Textes) interpretiert werden sollen (deswegen Maskierung):

```
"ich lerne \"PHP\"";
```

Wenn Sie mit HTML vertraut sind, wissen Sie, dass hier oft Anführungszeichen gesetzt werden/gesetzt werden sollen, denn Sie schreiben ja beispielsweise `<a name="oben">`. Diese Anführungszeichen müssen natürlich auch maskiert werden, wenn sie Teil einer PHP-Anweisung sind. Das sieht dann so aus:

```
echo "<a name=\"oben\">";
```

Wenn nun tatsächlich ein Backslash vor einem Anführungszeichen angezeigt werden soll, brauchen Sie einen zweiten:

```
echo "c:\\programme\\";
```

Dies zeigt das Abbildung 3.3. Sie sehen die Eingabe im Editor und die Ausgabe im Browser.

Bild 3.3: PHP erwartet einen zweiten Backslash

Auch für Zeilenumbrüche in PHP (nicht zu verwechseln mit dem
-HTML-Tag) gibt es ein Zeichen mit einem Backslash: \n. Damit wird ein Zeilenumbruch im von PHP erzeugten HTML-Quellcode ausgegeben.

Die folgende Tabelle bietet einen Überblick über die Zeichen, die Sie in doppelten Anführungszeichen verwenden können, damit bestimmte Zeichen ausgegeben werden.

Zeichenfolge	Effekt
\n	Neue Zeile
\r	Wagenrücklauf
\t	Horizontaler Tabulator
\\	Backslash
\$	Dollarsymbol
\" bzw. \'	Doppelte Anführungszeichen bzw. einfache Anführungszeichen

Tabelle 3.2: Ein Überblick über die Zeichen, die zur Ausgabe spezieller Zeichen verwendet werden

3.7 Kommentare

So lange man nur ein paar kleine Probescripts schreibt, fällt es schwer, sich vorzustellen, wie komplex – und mitunter ausufernd lang – ein Script sein kann. Wenn dann noch die nicht ungewöhnliche Situation auftritt, dass man sich Monate nach der ursprünglichen Programmierung erneut an den Programmcode setzt, um irgendetwas zu ändern oder zu verbessern, wäre man ohne erklärende Kommentare im Script mitunter verloren. Haben Sie hingegen mit aussagekräftigen Kommentaren gearbeitet, können Sie sehr viel einfacher rekonstruieren, was jeweils angewiesen wurde und warum Sie den Code so und nicht anders geschrieben haben.

Kommentare im PHP-Code werden nicht übertragen. Der Parser, der die Befehle ausführt, ignoriert die Zeile(n) einfach. Sie können Kommentare also auch getrost als »Notizzettel«, die nur für Sie selbst gedacht sind, nutzen. Es gibt zwei Methoden, eine Programmzeile im Script zu kommentieren. Entweder Sie schreiben // oder # an den Anfang der Zeile, die lediglich ein Kommentar sein soll. Das sieht beispielsweise so aus:

```
//Fehlermeldung zusammenbauen

if(!$name){$fehler="Bitte geben Sie einen Namen ein <br>";}
```

Um eine Programmzeile direkt zu kommentieren, können Sie den Kommentar auch an das Ende der Zeile schreiben:

```
$name=""; // löscht den Wert der Variablen
```

Benutzen Sie am Anfang des Kommentarteils das Zeichen `/*` und am Ende `*/`, wird der PHP-Prozessor alles ignorieren, was zwischen diesen Zeichen steht, ob nur eine Zeile oder auch mehrere.


HINWEIS

In einigen Versionen von PHP wurden PHP-Endzeichen in diesen Kommentarbereichen nicht als Kommentare, sondern immer noch als Endzeichen interpretiert. In der uns vorliegenden Version 5.0.4 ist dieses Fehlverhalten abgestellt.

```
<?php

/*
echo "<b>Heute ist Montag, willkommen auf unserer Seite</b>";
*/

?>
```

Würden Sie diesen Code speichern und die Datei im Browser aufrufen, würden Sie eine leere Seite sehen, weil ja nichts ausgegeben wird. Deswegen ersparen wir Ihnen hier ein Bild!

Kommentierungszeichen lassen sich auch hervorragend bei der Fehlersuche einsetzen. Wenn es Probleme mit einer Code-Zeile gibt, müssen Sie diese Zeile nicht gleich frustriert löschen und anschließend wieder neu eingeben, sondern Sie können sie einfach »auskommentieren«:

```
// if ($alter > 18); {
```


TIPP

Manche Editoren verwenden für Kommentare eine andere Farbe als die, die sonst im Script benutzt wird: Dies ist, wie Sie sich denken können, vor allem bei langen Scripts sehr hilfreich.

3.8 Variablen

Ein entscheidender Baustein in PHP-Scripts sind Variablen, mit denen Sie im Prinzip ständig arbeiten. Sie deklarieren (so heißt es professionell) Variablen im Script. Man könnte Variablen beschreiben als Behälter für Daten. Sie speichern temporär (während der Laufzeit des Scripts) die Daten bzw. Werte, die ihnen zugewiesen werden. (Technisch gesehen verhält es sich noch ein bisschen anders: Die Namen von Variablen verweisen auf einen bestimmten Speicherplatz im Rechner, an dem der Inhalt der Variable gespeichert ist.)

Eine bereits definierte Variable kann mit anderen Daten kombiniert und unter Beibehaltung des Ursprungswerts ergänzt und erweitert werden. Variablen sind also von Natur aus sehr flexibel oder eben: variabel! Sobald die Variable deklariert wurde, ist sie »einsatzfähig«.

3.8.1 Syntax und Wertezuweisung

Der Name einer Variablen ist frei wählbar. Die Namenskonvention erlaubt Buchstaben, Zahlen und Unterstriche. Der Name darf allerdings keine Leerstellen oder nicht-alphanumerischen Zeichen enthalten.

Ein gültiger Variablenname beginnt mit einem Buchstaben oder einem Unterstrich. Entscheidend ist: Allen Variablennamen ist ein Dollarzeichen vorangestellt. Der Wert wird nach einem Gleichheitszeichen (Zuweisungsoperator) zugewiesen und das Semikolon markiert das Ende. Werte, die aus Strings bzw. Zeichenketten (eine genauere Erklärung zu Zeichenketten finden Sie im Abschnitt *Datentypen*) bestehen, werden in Anführungszeichen gesetzt. Wenn Sie Zahlen zuweisen, dürfen keine Anführungszeichen verwendet werden, tun Sie es doch, werden Zahlen wie Text behandelt, sodass man nicht mit ihnen rechnen könnte. (So die Theorie, aber PHP ist »programmiererfreundlich« und ändert den Variablentyp bei Berechnungen wenn möglich in das benötigte Format.) Wenn mit Zahlen nicht gerechnet werden soll, dann werden sie wie Strings in Anführungszeichen gesetzt.

Ein paar Beispiele für Variablennamen und die Wertezuweisung:

```
$vorname = "Daniel";
```

Dieser Variablen wurde der Wert *Daniel* zugewiesen. Schreiben Sie in die nächste Zeile im Script

```
echo $vorname;
```

wird im Browser ausgegeben:

```
Daniel
```

Weitere Beispiele für gültige Variablennamen:

```
$nummer1 = 100;
$nummer_1 = 100.00;
$_Text1 = "Last Minute";
$jahr_alt1 = "2005";
```

Im Regelfall werden Variablen jeweils mit dem neuen, in der Reihenfolge des Scripts zuletzt zugewiesenen Wert überschrieben (man spricht auch von: changed on the fly). Passen Sie auf:

```
$beispiel = "hier dreht es sich um Variablen";
$beispiel = "wir behandeln Datentypen";
```

Geben Sie an dieser Stelle mit `echo` den Wert der Variablen `$beispiel` aus, wird der letzte Satz gedruckt bzw. angezeigt werden.

Sie sehen dies in Abbildung 3.4.

Variablennamen unterscheiden zwischen Groß- und Kleinschreibung (sie sind *case-sensitive*). `$Nummer` und `$nummer` wären also nicht die gleiche Variable. Erfahrungsgemäß empfiehlt es sich, grundsätzlich alles kleinzuschreiben, dann müssen Sie sich nicht daran erinnern, ob Sie irgendeinen Teil des Variablennamens groß- oder kleingeschrieben haben, bzw. das Script mühselig danach durchforsten. Ein Name wie `$gartenHaus_nr3` wäre nicht besonders klug gewählt, denn mit Sicherheit wissen Sie irgendwann nicht mehr, dass Sie diese merkwürdige Groß- und Kleinschreibung verwendet haben. Außerdem ist es ratsam, mehr oder minder aussagekräftige Namen zu verwenden, die Ihnen auch später noch ihre Bedeutung verraten. Hüten Sie sich vor kryptischen Abkürzungen.

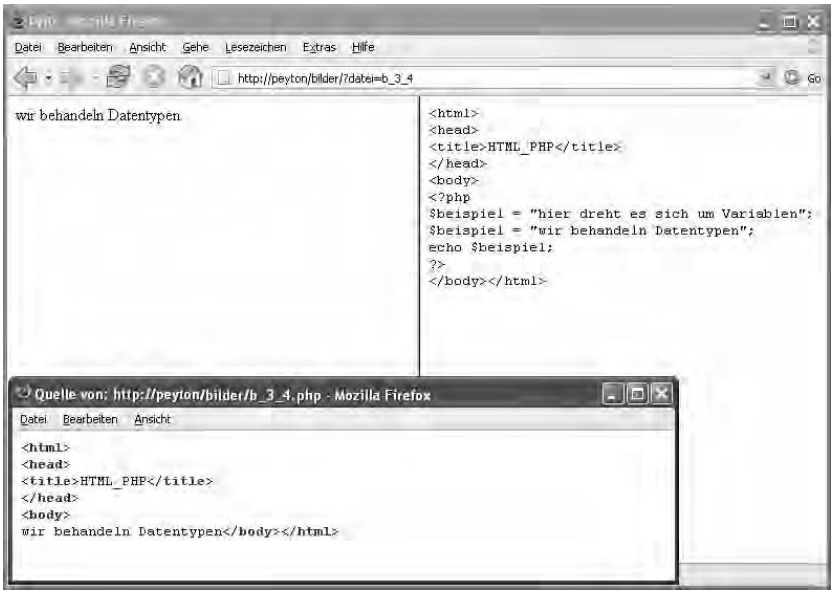

Bild 3.4: Variablen werden überschrieben

Bei der Ausgabe des Inhalts einer Variablen mit dem Befehl `echo` schreiben Sie den Variablennamen nicht in Anführungszeichen, sie würden aber auch nicht stören. Wenn Sie aber einen Text durch mehrere Variablen ausgeben lassen, werden Anführungszeichen benötigt:

```
$stadt = "Berlin";
$code = 10666;
echo "$stadt <br>";
echo "$code $stadt";
```

Abbildung 3.5 zeigt den Code und die Ausgabe (achten Sie auch auf das `<br>` für den Zeilenumbruch; als Teil des PHP-Codes muss es mit `echo` ausgegeben werden!)

Wertezuweisung durch Variablen

Variablen können ihren Inhalt auch von anderen Variablen erhalten. Das sieht ganz ähnlich aus, aber die Anführungszeichen entfallen:

```
$testvar = $testvariable1;
```

Beachten Sie bitte: Wenn eine Variable ihren Inhalt von einer anderen Variablen erhalten hat, ist sie hartnäckig und besteht auf diesem Inhalt, auch wenn der Inhalt der Ursprungsvariablen geändert wird. Um dies noch einmal deutlich zu machen, schauen Sie sich das unten stehende kleine Scriptfragment an:

```
$variable1 = "Hans";
$variable2 = $variable1;
echo $variable2;
```

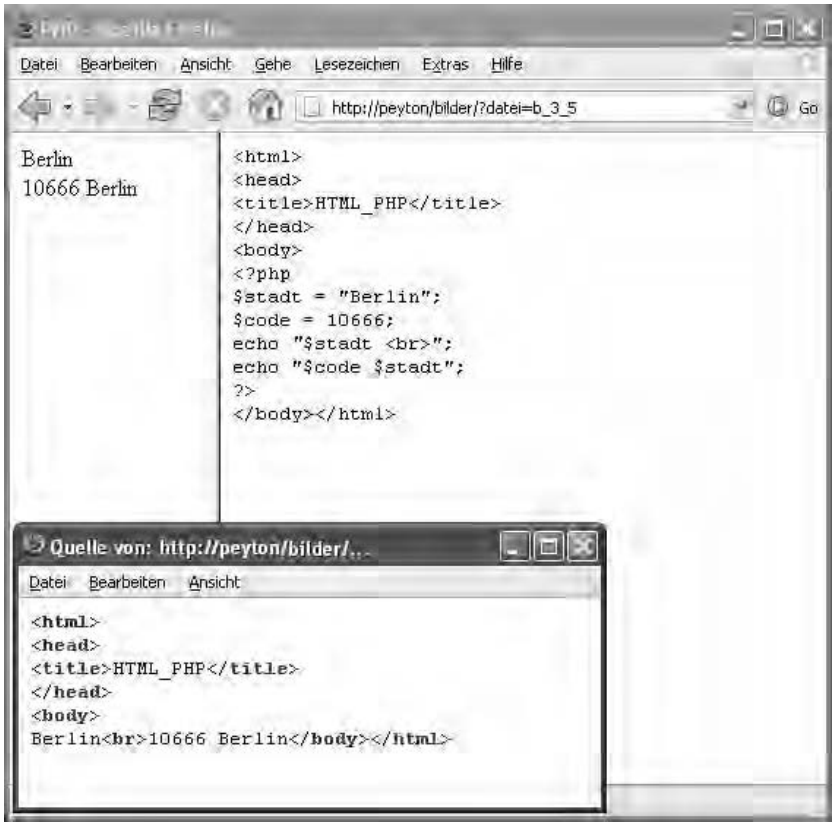

Bild 3.5: Die Inhalte von Variablen ausgeben

Ausgabe im Browser: Hans

```
//Verändern der Variablen $variable1
$variable1 = "Daniel";
echo $variable2;
```

Ausgabe im Browser: Hans

Allerdings bietet PHP nun auch eine Wertezuweisung durch Referenzierung. In dem Fall zeigen beide Variablen auf die gleiche Speicherstelle mit dem Effekt, dass Änderungen der neuen Variablen auch die Ursprungsvariable ändern und umgekehrt. Für die Zuweisung per Referenz wird der Ausgangsvariablen ein `&` vorangestellt.

```
$tag = "Montag";
$tagneu = &$tag;
$tag = "Dienstag";
echo $tagneu;
```

Ausgabe im Browser: Dienstag

Abbildung 3.6 zeigt noch einmal beides zusammen, den Code und die Ausgabe.

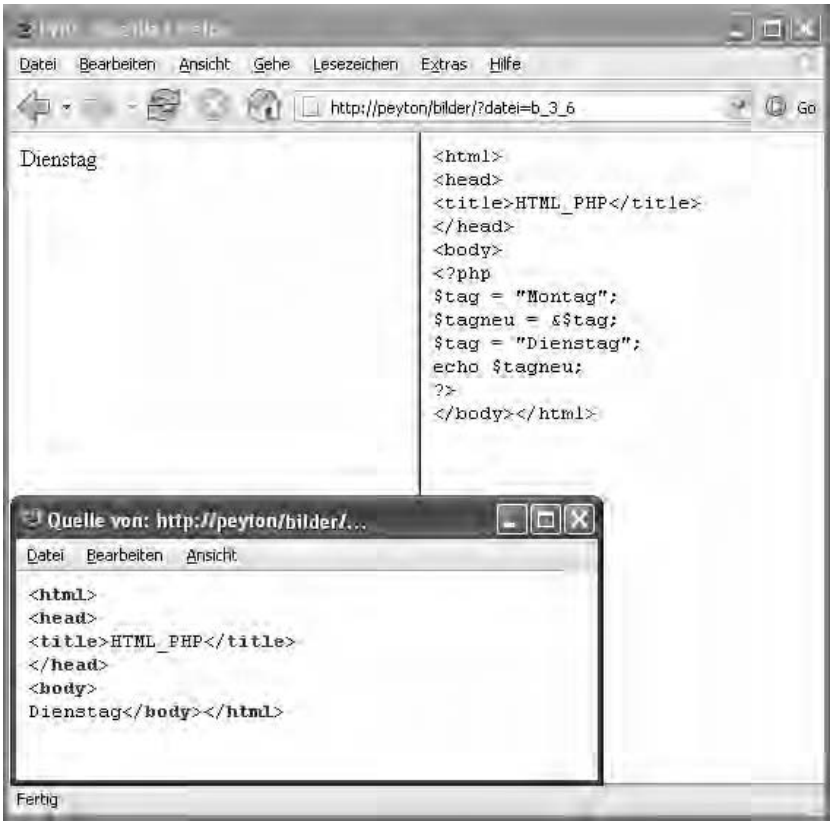

Bild 3.6: Zuweisung von Variablen per Referenzierung

String-Variablen erweitern

String-Variablen lassen sich erweitern/verlängern. Dies funktioniert durch den Verkettungsoperator, dargestellt durch einen Punkt. Bei dieser Methode verwenden Sie den- bzw. dieselben Variablennamen und bei der Zuweisung des nächsten Wertes einen Punkt vor dem Gleichheitszeichen.

```
$name  = "der Vorname ist ";
$name .= "Daniel!";
```

In der Variablen steht nun *der Vorname ist Daniel!*. Geben Sie die Variable mit `echo $name;` aus, lesen Sie im Browser:

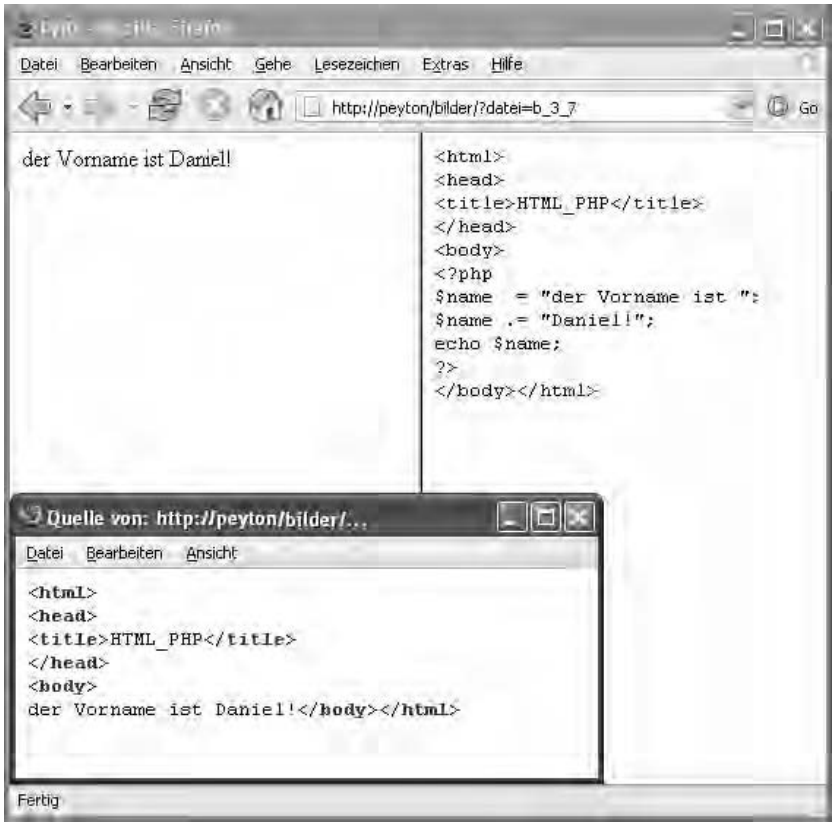

Bild 3.7: Die Ausgabe im Browser

Hätten wir den Punkt weggelassen, wäre der erste Wert der Variablen mit dem zweiten Wert (Daniel!) überschrieben worden. Die Erweiterung einer Variablen funktioniert nicht nur einmal, sondern quasi so oft Sie möchten. Ergänzen Sie das Script um:

```
$name .= " Er ist 17";
```

wird bei der Ausgabe der Variablen `$name` auch dieser Satz zu lesen sein. Achten Sie bei Erweiterungen auf Leerstellen, die natürlich innerhalb der Anführungszeichen stehen müssen, ansonsten klebt der Text beieinander. In Abbildung 3.8 sehen Sie den Code und die Ausgabe.

String-Variablen verketten

Etwas anders funktioniert die Verkettung. Dabei wird der Punkt verwendet und – technisch gesprochen – eine Zeichenkette ausgegeben, die aus dem rechten und linken Operand zusammengesetzt ist. Unabhängig vom Datentyp werden bei dem Einsatz des Punktes als Verkettungsoperator die Operanden als Zeichenkette behandelt. Die Syntax sieht folgendermaßen aus:

```
$nr1 = "aha";
$nr2 = $nr1." eine Verkettung";
```

Achten Sie hierbei auf die Leerstelle vor dem Wort *eine*, damit auch bei der Ausgabe dort eine Leerstelle ist, sonst hieße es *ahaeine Verkettung*, was nicht Sinn der Sache ist!

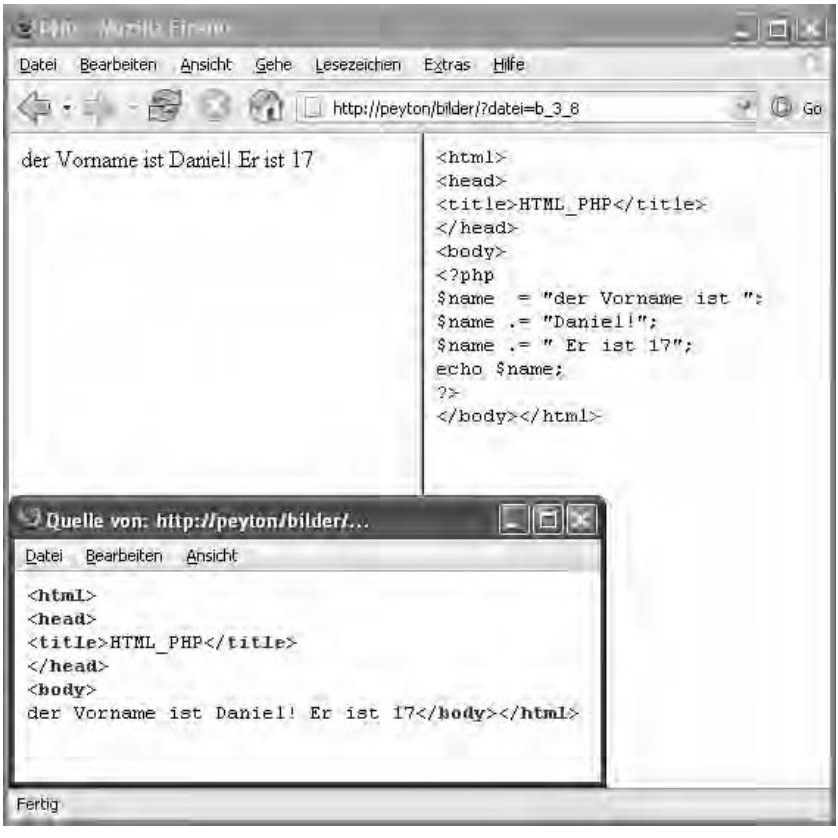

Bild 3.8: Variablen können erweitert werden

Abbildung 3.9 zeigt den Code und die Ausgabe im Browser.

Eine derartige Verkettung kann weiterentwickelt werden. Die Crux sind zum Teil die Leerstellen, achten Sie gut darauf, dass sie (als Zeichenkette) bei einer Verkettung auch in Anführungszeichen gesetzt werden müssen. Werfen Sie einen Blick auf Abbildung 3.10. Durch eine weitere Verkettung wird der Variablen `$nr4` ihr Wert zugewiesen.

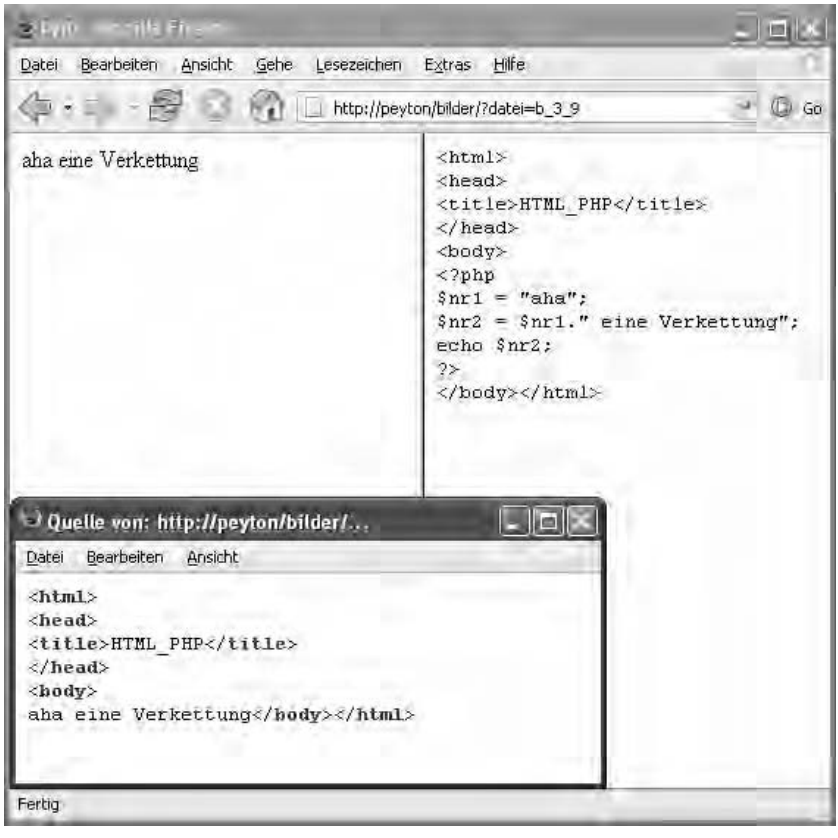

Bild 3.9: Variablen und Text können verkettet werden

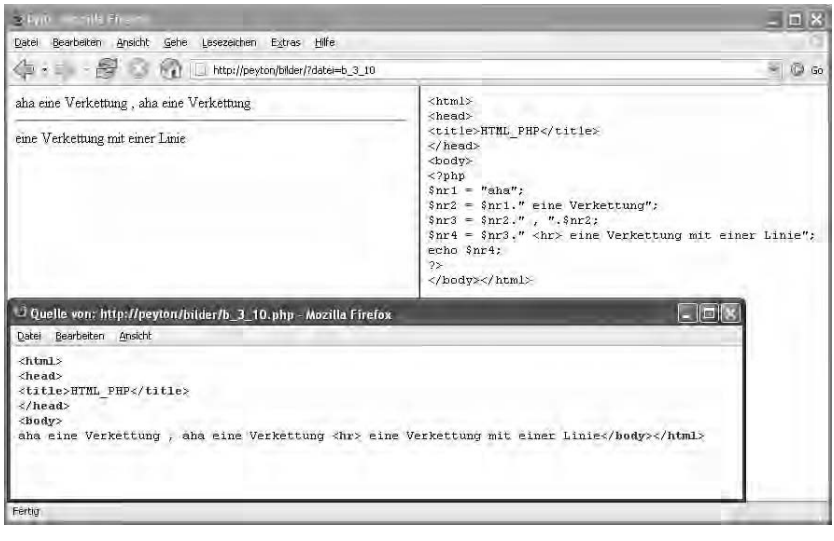

Bild 3.10: Noch mehr Elemente in der Verkettung


HINWEIS

Statt die Werte von Variablen im Script festzulegen, können Sie auch übergeben werden, beispielsweise durch User-Eingaben in ein Formular, was in der Praxis natürlich ganz oft der Fall ist. In dem Fall wird mit eindeutigen Namen gearbeitet, d.h. die Formularfelder erhalten einen eindeutigen Namen und dieser Name wird bei der Auswertung als Arrayelement in `$_POST` bzw. `$_GET` verwendet. Darauf wird später natürlich ausführlich eingegangen.

Die Existenz von Variablen prüfen

In nicht wenigen Situationen ist es wichtig zu prüfen, ob eine Variable mit einer leeren Zeichenkette gefüllt ist (oder mit 0) oder überhaupt zugewiesen wurde. Es gibt eine Funktion, die es Ihnen ermöglicht, Variablen auf ihre Existenz hin zu prüfen. Diese Funktion lautet:

```
isset($variable)
```

Zu gut Deutsch also etwa: ist gesetzt. In der Klammer wird die Variable erwartet, deren Existenz Sie überprüfen möchten. Die Überprüfung gibt *true* (`wahr`) zurück, wenn die Variable existiert. Auch wenn wir mit dem folgenden kleinen Scriptfragment etwas vorgreifen (da eine `if`-Anweisung verwendet wird), soll der Gebrauch von `isset()` hier schon mal kurz demonstriert werden:

```
<?php

$foo = "test";
if (isset($foo))
{ echo "foo existiert"; }
else
{ echo "foo existiert nicht"; }

?>
```

Umgekehrt kann man eine Variable auch wieder ungültig machen, was – wie Sie später in den Beispielen dieses Buches sehen werden – mitunter notwendig ist. Die entsprechende Funktion lautet

```
unset($variable)
```

Ob einer Variablen ein Wert zugewiesen wurde, lässt sich ebenfalls überprüfen:

```
empty($variable)
```

Hier erhalten Sie als Ergebnis der Prüfung `wahr`, wenn der Inhalt der Variablen 0 oder eine leere Zeichenkette ist.

Dynamische Variablen

Unter dynamischen Variablen versteht man die Möglichkeit, Variablennamen in Variablen zu speichern. Hier müssen Sie ein bisschen »um die Ecke« denken. Es verhält sich folgendermaßen: Die dynamische Variable nimmt den Wert der zuvor definierten gleichnamigen »normalen«

Variablen als ihren Namen. Mit zwei vorangestellten Dollarzeichen kann man einer dynamischen Variablen einen Wert zuweisen. In einem Script kann das z.B. so aussehen:

```
$vari = "Vorname";
$$vari = "Daniel";
echo $Vorname;
```

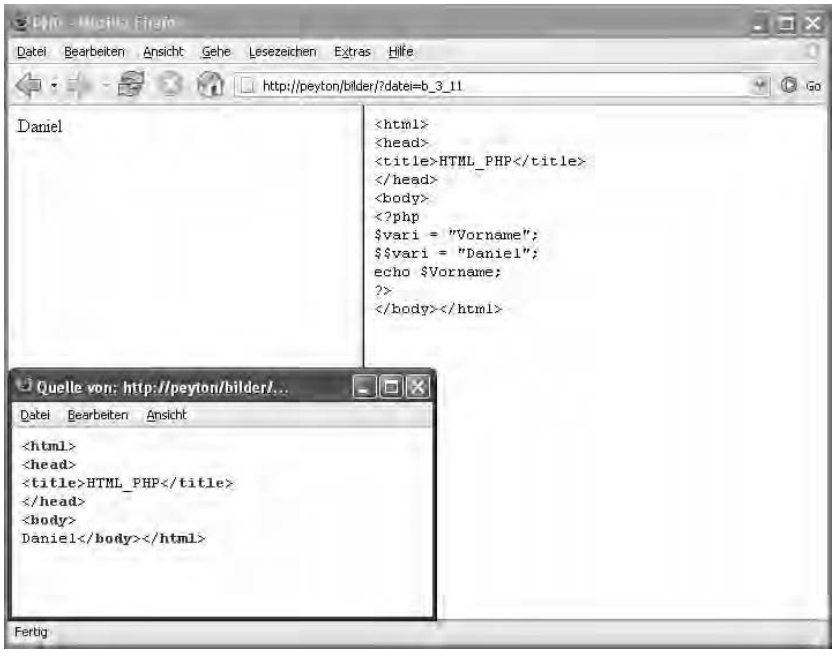

Bild 3.11: Die Wirkungsweise dynamischer Variablen

3.9 Datentypen

Wie Sie oben an den paar Beispielen bereits gesehen haben, können Sie Variablen Werte zuweisen, die unterschiedlichen Datentypen zuzuordnen sind. PHP erkennt den Datentyp bei der Zuweisung automatisch und versucht, den Wert dann entsprechend zu behandeln.

Insgesamt gibt es acht Daten-Typen: strings, integers, floats, booleans, arrays, objects, resources und NULL. Da man in der Regel mit allen Typen zu tun hat, kann es nicht schaden zu wissen, welche Art von Information die einzelnen Typen enthalten. Daher finden Sie in der Tabelle eine kurze Übersicht.

Variablen-Typ	Inhalt
string	Zeichen in fast unbegrenzter Anzahl
integer	ganze (positive und negative) Zahlen
float	Dezimalzahlen
boolean	true oder false (wahr/unwahr). PHP betrachtet einen leeren String, ein Array ohne Elemente und NULL als false, alles andere als true.
array	eine Anzahl von Werten
object	komplexe Variablen
resource	Daten, die nicht aus PHP stammen, z.B. ein Bild, das Resultat einer Abfrage, die Verbindung zur Datenbank etc.
NULL	kann nur den Wert NULL annehmen. Eine Variable gilt als NULL, wenn NULL oder bisher kein Wert zugewiesen wurde, oder die Variable mit `unset` gelöscht wurde.

Tabelle 3.3: Die verschiedenen Datentypen

Wie oben bereits erwähnt, werden Datentypen vom PHP-Interpreter konvertiert, wenn es der Sache dienlich scheint. Schauen Sie sich dieses Stückchen Code an; es demonstriert, dass PHP relativ »locker« mit Datentypen umgehen kann.

```
<?php

$zeichen = "12";
$integer = 12;
echo $zeichen + $integer;

?>
```

Als Resultat würden Sie 24 erhalten, obwohl `$zeichen` einen String enthält (wegen der Anführungszeichen). PHP verwandelt den String in den Datentyp integer, da es clever genug ist zu erkennen, dass vermutlich eine Rechenoperation durchgeführt werden soll. Mitunter gibt es natürlich auch Probleme beim Umwandeln, z.B. wenn ein boolean, der auf *false* gesetzt ist, in ein String verwandelt wird.

```
<?php

$bool = true;
print "Bool ist $bool";
$bool = false;
print "Bool ist $bool";

?>
```

Als Ausgabe würde im ersten Fall 1 erscheinen, im zweiten Fall nichts (leerer String). Hier müsste man PHP dann einfach genauer mitteilen, was es machen soll und schreiben:

```
$bool = false;
print "Bool ist ";
print (int)$bool;
```

Mit diesen Zeilen würden Sie dann für `false` 0 erhalten.

3.9.1 Strings oder Zeichenketten

In dem obigen Beispiel

```
$vorname = "Daniel";
```

enthält die Variable eine Zeichenkette oder einen – dies ist die andere Bezeichnung für Zeichenketten – String. Es handelt sich also um eine Variable vom Typ String. Von einem String spricht man bei Eingaben, die aus Buchstaben oder aus einer Kombination von Buchstaben, Zahlen (mit denen keine Rechenoperation durchgeführt wird), Symbolen und Leerstellen bestehen und sich zwischen einfachen oder doppelten Anführungszeichen befinden. Strings wären beispielsweise:

```
"Hallo Welt"
"Hallo, $vorname"
"1999"
"Haus 59"
```

Folgendes ist wichtig: Im Zusammenhang mit dem Befehl `echo` spielt es eine Rolle, ob Sie für die Ausgabe doppelte oder einfache Anführungszeichen gebrauchen. Der Unterschied ist der, dass bei doppelten Anführungszeichen die Variablen mit ausgewertet (geparst) werden. Würden Sie stattdessen `echo 'Hallo, $vorname';` schreiben, also einfache Anführungszeichen benutzen, würde `'Hallo, $vorname'` ausgegeben werden. Das ist meistens nicht Sinn der Sache! Soll tatsächlich ein Dollarzeichen innerhalb von doppelten Anführungszeichen ausgegeben werden – im folgenden Beispiel `$dollar` – können Sie das Zeichen auch mit dem Backslash maskieren:

```
echo "\$dollar";
```

Strings besitzen eine Eigenschaft, die sie von anderen Datentypen abheben, und zwar die {}-Schreibweise – auch als komplexe Syntax bezeichnet. Man kann durch geschweifte Klammern nach dem String auf ein gewünschtes Zeichen zugreifen (und es gegebenenfalls modifizieren). Werfen Sie einen Blick auf den Code-Schnipsel:

```
<?php

$meintext="lext mit klammer?";
$meintext{0}="t";
$meintext{16}="!";
print $meintext;

?>
```

Die `0` in der ersten geschweiften Klammer greift auf das erste Zeichen zu und wird ausgetauscht durch ein »t«, die `16` schnappt sich das letzte Zeichen des Strings und ein Ausrufezeichen wird in die Variable geschrieben.

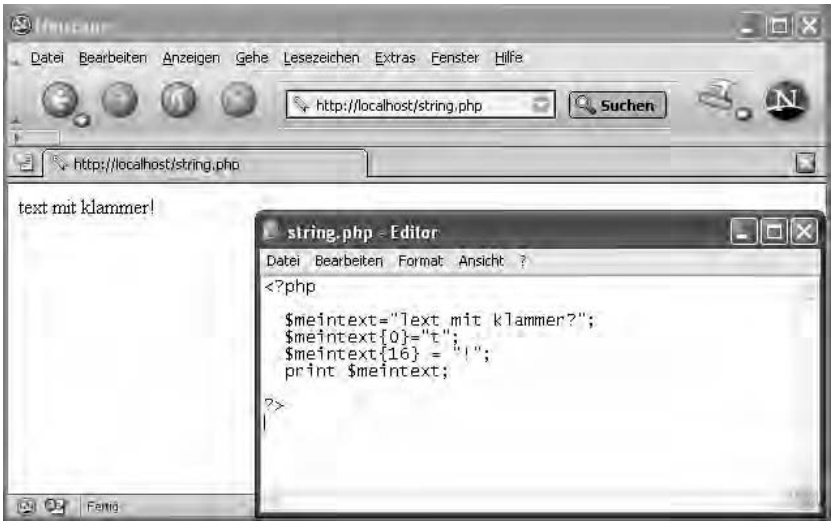

Bild 3.12: Komplexe Syntax mit {} bei Strings

Zeichenketten lassen sich in vieler Hinsicht manipulieren und bearbeiten. Ein paar Funktionen wurden bereits weiter oben erwähnt. Im Kapitel über vordefinierte Funktionen und im Lauf der nächsten Kapitel, in denen wir konkrete Projekte realisieren, werden Sie weitere kennen lernen.

3.9.2 Zahlen

Bei dem Datentyp Zahl müssen Sie unterscheiden zwischen Integer oder Ganzzahl (eine Zahl ohne Nachkommastellen) und Double oder Fließkommazahl, auch als Float bezeichnet.

Als Integer können beispielsweise gelten:

- 10
- -10

Beispielwerte für den Typ Double wären:

- 10.50;
- -10.50;

Achten Sie darauf, dass Sie als Dezimalstelle den Punkt nehmen und nicht ein Komma, was uns ja im Prinzip geläufiger ist.

Werte inkrementieren

Im Verlauf der folgenden Kapitel (und speziell im nächsten Kapitel, das vordefinierte Funktionen unter die Lupe nimmt) werden Sie in unterschiedlichen Zusammenhängen erfahren, wie Sie Zahlen bearbeiten und ihnen z.B. ein bestimmtes Format zuweisen können. An dieser Stelle wollen wir Sie in einen Vorgang einweihen, der in Scripts wirklich sehr häufig vorkommt. Es geht darum, den Wert einer Variablen jeweils um eins hochzuzählen, eine Aktion, die als Inkrementieren bezeichnet wird.

Denkbar ist das folgende Vorgehen bzw. die folgende Schreibweise:

```
$zahl = 0;
$zahl = $zahl + 1;
```

Praktischer ist die Kurzform, die auch meistens verwendet wird. Sie können einfach zwei Pluszeichen an den Wert hängen:

```
$zahl++;
```

Wenn Sie diese Aktion in einem Script probieren möchten, lassen Sie die Zahlen jeweils ausgeben. Wir geben noch einmal genau vor, was zu tun ist:

1. Öffnen Sie gegebenenfalls Ihren Editor und schreiben Sie das PHP-Startzeichen `<?php`.
2. Setzen Sie die Variable `$zahl=0;`.
3. Lassen Sie mit dem Befehl `echo` den Wert ausgeben.
4. Für bessere Lesbarkeit weisen Sie mit `echo "<br>";` einen Zeilenumbruch an.
5. Nun lassen Sie den eben gesetzten Wert 0 um eins hochzählen: `$zahl++;`
6. Beenden Sie PHP, speichern Sie das kleine Script als PHP-Datei und testen Sie es im Browser.

Um das Gegenteil zu erreichen, also um zu dekrementieren, schreiben Sie statt der zwei Pluszeichen einfach zwei Minuszeichen, beispielsweise:

```
$zahlneu=10;
$zahlneu--;
```

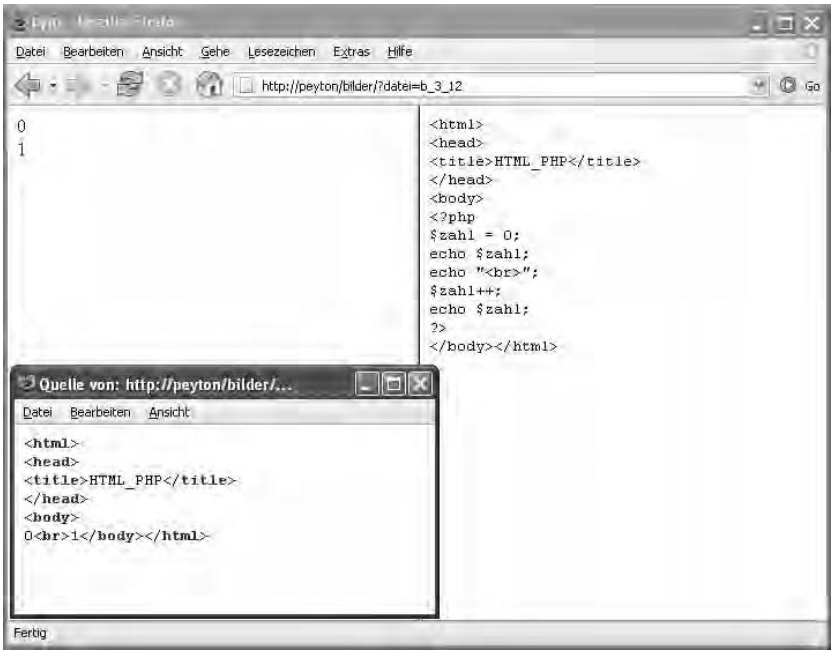

Bild 3.13: Zahlen mit ++ inkrementieren


HINWEIS

Sie können die doppelten Plus- bzw. Minuszeichen auch vor den Variablennamen schreiben. Die Position führt zu unterschiedlichem Verhalten bei PHP. Bei Zuweisungen wird mit `$neuezahl = $zahl++;` zunächst der Wert von `$zahl` der Variablen `$neuezahl` zugewiesen und dann um eins erhöht. In `$neuezahl` steht dann z. B. 4 und in `$zahl` 5. Schreiben Sie hingegen `$neuezahl = ++$zahl;`, wird `$zahl` erst inkrementiert und dann `$neuezahl` zugewiesen, sodass in `$zahl` und `$neuezahl` nach der Zeile die gleichen Werte stehen.

Arithmetische Operatoren

Im Zusammenhang mit dem Datentyp Zahlen werden Sie in Scripts häufig arithmetische Operatoren verwenden. Dies sind auch in PHP die üblichen Operatoren, die Sie für grundlegende Berechnungen einsetzen:

Operatoren	Rechenart
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

Tabelle 3.4: Die Zeichen für arithmetische Operatoren

Sie könnten also beispielsweise folgende Variable setzen:

```
$kosten = 12 * 50;
```

Wenn Sie dann schreiben würden:

```
echo "Sie zahlen im Jahr Euro $kosten";
```

hieße es auf der Webseite:

> Sie zahlen im Jahr Euro 600

Abbildung 3.14 zeigt beides, das Script im Editor und die Ausgabe im Browser.

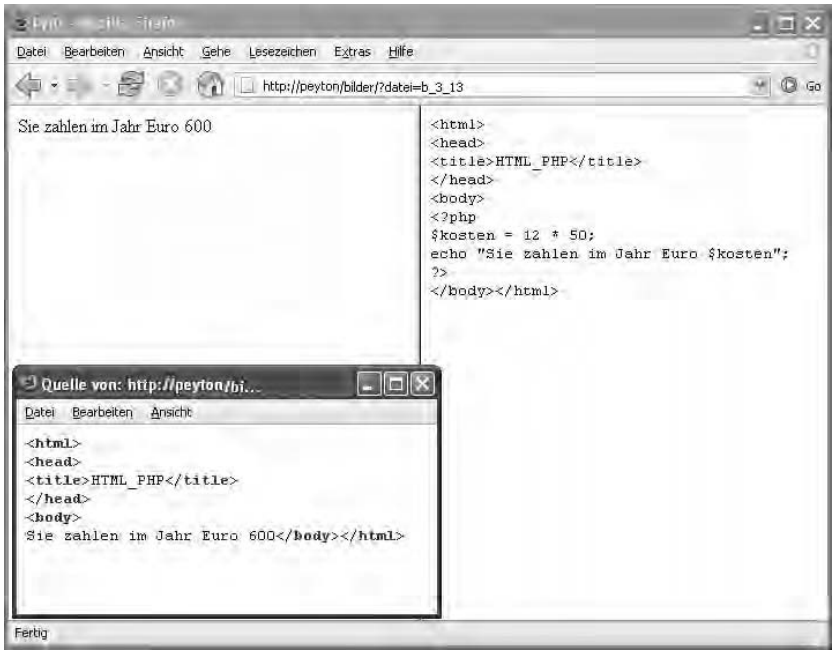

Bild 3.14: Für Berechnungen setzen Sie die üblichen Operatoren ein

Eine geschicktere Variante wäre es, vorher andere Variablen zu deklarieren und dann die Berechnung mithilfe dieser Variablen erfolgen zu lassen:

```
$monat = 12;
$beitrag = 50;
$kosten = $monat * $beitrag;

echo "Sie zahlen im Jahr Euro ".$kosten;
```

Der Vorteil: Ihr Script ist einfach flexibler. Ändert sich beispielsweise der Beitrag, brauchen Sie lediglich den Wert der entsprechenden Variablen zu ändern.

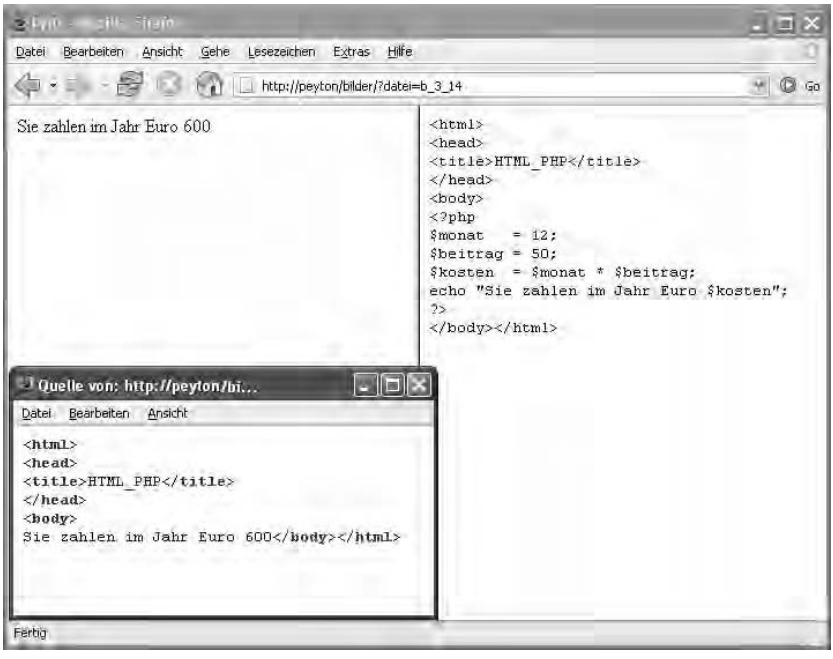

Bild 3.15: Eine einfache Berechnung – definieren Sie Variablen, um ein Script flexibel zu halten

Präzedenz

PHP rechnet übrigens auf »normale« Art und Weise, d.h. folgt den üblichen mathematischen Regeln. Eine dieser Regeln besagt bekanntlich: Punktrechnung geht vor Strichrechnung. Diese Regel müssen Sie beachten, wenn Sie Variablen Werte zuweisen und dabei mehrere Operatoren benutzen. Unvermeidbar ist dann in bestimmten Fällen der Gebrauch von Klammern, die – wie Sie ja wissen – zuerst berechnet werden, also eine so genannte Präzedenz erzwingen. Sie müssen also beispielsweise, je nachdem, wie die Berechnung aussehen soll, schreiben:

`$kosten = (20 - 8)/4` oder aber `$kosten = 20 - (8/4)` bzw. `$kosten = 20 - 8/4`

3.9.3 Datentypen feststellen und festlegen

Um festzustellen, welcher Datentyp intern von PHP genutzt wird, können Sie eine Funktion benutzen, die den jeweiligen Typ zurückgibt. Die Funktion lautet

```
gettype()
```

In die Klammer schreiben Sie die Variable, deren Typ Sie sich anzeigen lassen möchten:

```
<?php
$datentyp=10;
echo gettype($datentyp);
echo "<br>";
$datentyp="Hallo";
echo gettype($datentyp);
?>
```

Der Browser zeigt damit untereinander (wegen `<br>`) die Datentypen integer und string, zu erkennen in der Abbildung.

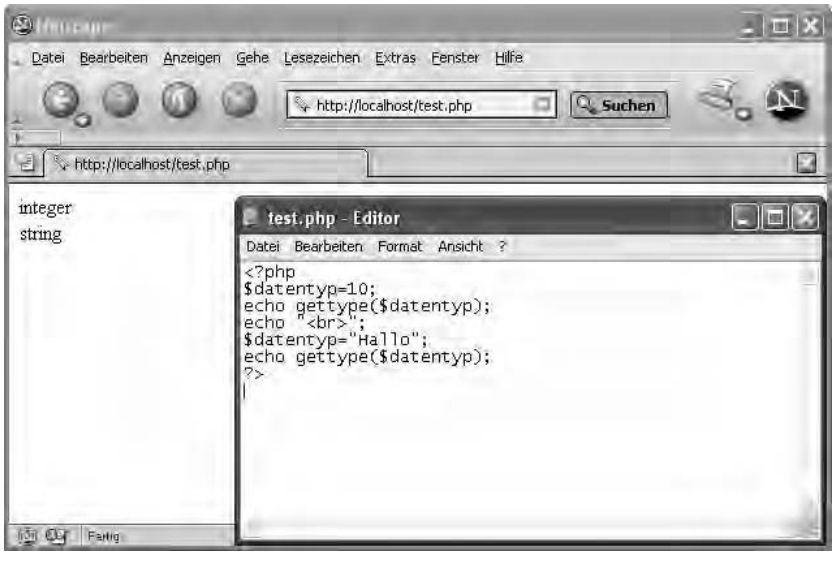

Bild 3.16: Datentypen anzeigen lassen

Das Gegenstück zu `gettype()` ist die Funktion `settype()`. Damit kann der Typ einer Variablen geändert werden, was mitunter notwendig ist. Die Klammer erwartet als Argumente die zu ändernde Variable und – durch Komma getrennt – den gewünschten Datentyp. Angenommen, es gibt die Variable `$zahl = 4711` vom Datentyp integer. Um daraus einen String zu machen, würden Sie schreiben:

```
settype($zahl, "string");
```

Mit der Funktion `gettype()` (und `echo`) können Sie überprüfen, ob die Konvertierung geklappt hat.

Manuelle Konvertierung

PHP bietet noch eine andere Methode, einen Datentyp bewusst festzulegen, also unter Umständen auch zu erzwingen. Die Schreibweise ist einfach, der Datentyp wird direkt in Klammern vor die Variable gesetzt, die umgeformt werden soll:

```
<?php

$meinstring ="schöner text";
$meininteger = (integer)$meinstring;

?>
```

Zunächst enthält – wie Sie sehen – `$meinstring` eine Zeichenkette. Durch das `integer` in der Klammer wird PHP dazu überredet, daraus einen integer zu machen.

Diese Art der Konvertierung, die auch als Type Casting bezeichnet wird, eignet sich übrigens auch ganz prima zum Abrunden von Zahlen, da Sie einen float (also eine Dezimalzahl) ganz einfach in einen integer wandeln können. Dabei wird automatisch abgerundet. Werfen Sie einen Blick auf die Abbildung, die das kleine Script und die Ausgabe zeigt:

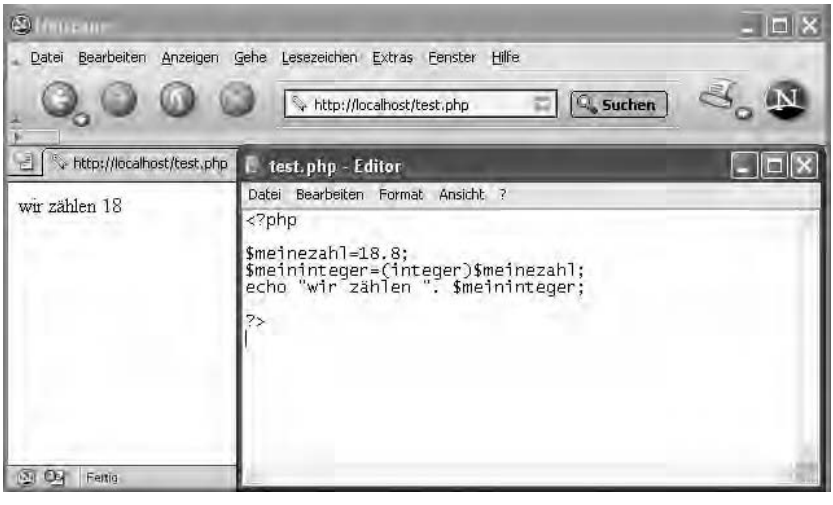

Bild 3.17: Aus float wird ein integer

3.10 Die Welt der Konstanten

Weiter oben haben wir uns mit Variablen beschäftigt und deutlich zu machen versucht, dass Variablen zum A und O beim Programmieren mit PHP gehören. Wir machen Sie nun mit Konstanten bekannt, die prinzipiell auch Variablen sind.

Wenn Ihnen auffällt, dass Sie beim Schreiben eines Scripts eine Variable deklariert haben, die sich während des ganzen Scripts nicht ändert, hätten Sie mit einer Konstanten unter Umständen eine bessere Wahl getrof-

fen. Wie der Name schon suggeriert, kann sich der Wert einer Konstanten zur Laufzeit eines Scripts nicht ändern, er bleibt »konstant«. Einmal definiert, können Konstanten nicht gelöscht werden.

Konstanten sind bei der Verarbeitung ein kleines bisschen schneller als Variablen. Ihnen wird kein Dollarzeichen vorangestellt; damit heben sie sich also optisch gut von Variablen ab. Außerdem sind sie im Gegensatz zu Variablen automatisch global, müssen also in lokalen Bereichen und Namensräumen nicht extra »globalisiert« werden (lesen Sie dazu am besten den Hinweis weiter unten!). Ihr Vorteil liegt also vor allem in ihrer Verfügbarkeit.


HINWEIS

Im Großen und Ganzen kann man zwischen zwei so genannten Geltungsbereichen unterscheiden: den globalen und den lokalen. Der globale Bereich eines Scriptes ist der komplette Bereich inklusive der Funktionen und Objekte. Lokale Bereiche dagegen sind Bereiche innerhalb von Funktionen, Objekten. Alle dort deklarierten Variablen sind auch nur dort verfügbar, während Konstanten per se »überall« gültig sind (das Ganze wird später, wenn Sie auch mit Funktionen/Objekten zu tun haben, wesentlich verständlicher!).

3.10.1 Syntax und Definition

Während Variablen ihr Wert einfach zugewiesen wird, werden Konstanten definiert.

Dazu verwenden Sie die Funktion

```
define(Name, Wert)
```

Wie schon gesagt, es wird kein Dollarzeichen vorangestellt. Ansonsten folgt die Bezeichnung den gleichen Regeln wie alle anderen Bezeichner in PHP. Ein gültiger Name beginnt mit einem Buchstaben oder einem Unterstrich, gefolgt von beliebig vielen Buchstaben, Zahlen oder Unterstrichen.

Nach gängiger Konvention werden Konstanten immer in Großbuchstaben geschrieben, wobei im Prinzip zwischen Groß- und Kleinschreibung unterschieden werden muss; Konstanten sind von Haus aus also casesensitive.

Konstanten könnten also heißen

TITEL, TITEL_NEU, TITEL3, aber nicht: 3TITEL

Konstanten können nur so genannte skalare Daten (boolean, integer, float und string) enthalten.


HINWEIS

Sie benutzen für die Definition einer Konstanten eine eingebaute Funktion von PHP; diese Funktionen werden ausführlich im nächsten Kapitel behandelt. Wir greifen hier also ein bisschen vor.

Betrachten wir nun die Definition einer Konstanten. In der Klammer werden zwei Angaben (Parameter gesetzt) gemacht, erstens der Name selbst – achten Sie auf die Anführungszeichen – und zweitens, durch Komma getrennt, der Wert, den sie erhält. Ein kleines Beispiel, in dem wir eine Konstante definieren, die als Wert den String `Magnum` erhält:

```
<?php'
define("ARTIKEL_NEU", "Magnum");
print "wir möchten das ".ARTIKEL_NEU;
?>
```

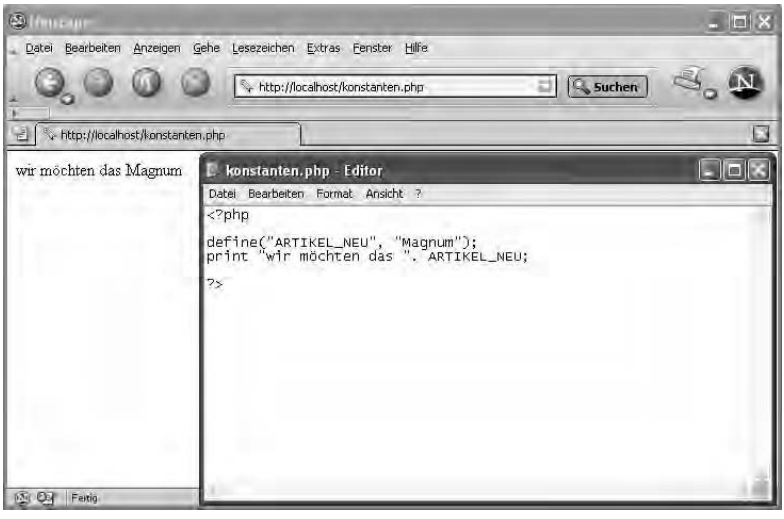

Bild 3.18: Konstanten werden definiert

`ARTIKEL_NEU` gibt also wie gewünscht Magnum aus. Achten Sie in diesem kleinen Script auch drauf, dass wir in der `print`-Zeile den Punkt als Verkettungsoperator benutzt haben. Wir verketten also den Text inklusive der Leerstelle und die Konstante, damit beides ausgegeben wird; täten wir das nicht, würde PHP uns mit einer Fehlermeldung erschrecken.

Es gibt eine Möglichkeit, die Sensibilität der Konstanten gegenüber Groß- und Kleinschreibung zu umschiffen. Wenn Sie die Parameter in der Klammer ergänzen durch *true* (wieder durch ein Komma abgetrennt), achtet PHP nicht mehr auf die Groß- und Kleinschreibung und das Script würde auch funktionieren, wenn Sie `artikel_neu` aufrufen würden; probieren Sie es aus:

```
define("ARTIKEL_NEU", "Magnum", true);
```

3.10.2 Umgang mit Konstanten

Im Zusammenhang mit Konstanten sind zwei drei weitere Funktionen hilfreich, die wir kurz erwähnen möchte (auch wenn – wie gesagt – das Thema »Funktionen« an dieser Stelle etwas verfrüht ist): `defined()`, `constant()`, `get_defined_constants()`

Die Funktion `defined()` benutzen Sie, um herauszufinden, ob eine Konstante definiert ist. Mit `constant()` lesen Sie den Wert einer Konstanten aus. Wenn man bedenkt, dass man diesen Wert auch erhält, indem man den Namen der Konstante einfach (mit `echo` oder `print`) ausgibt, erscheint `constant()` etwas überflüssig. Aber was, wenn Sie sich des Namens nicht sicher sind? Dann hilft `constant()`, wenn Sie die Konstante in eine Variable geschrieben haben:

```
$konstante = "ARTIKEL_NEU";
print constant($konstante);
```

Mit `get_defined_constants()` erhalten Sie eine Liste aller definierten Konstanten.

PHP kennt mit `__FILE__` und `__LINE__` vordefinierte Konstanten. `__FILE__` liefert den Namen der momentan ausgeführten Datei und `__LINE__` die Nummer der gerade bearbeiteten Zeile.

3.11 Arrays

Arrays sind Sonderformen von Variablen, denn in Arrays können beliebig viele Werte gespeichert werden. Sie stellen gewissermaßen eine Sammlung von Werten in einer Variablen dar. Arrays können Strings und Zahlen enthalten (oder andere Arrays). Die Namen von Arrays werden ebenfalls mit einem Dollarzeichen gebildet, gefolgt von einer eckigen Klammer. Ansonsten unterliegen die Namen der gleichen Namenskonvention wie andere Variablennamen.

Um Arrays zu verstehen, wird häufig der Vergleich mit einer Tabelle herangezogen. Werfen Sie einen Blick auf die folgende kleine Tabelle:

Index oder Schlüssel	Wert
1	Hosen
2	Röcke
3	Kleider
4	Jacken
5	Pullover

Tabelle 3.5: Arrays sind strukturiert wie Tabellen. Im Beispiel werden Zahlen als Schlüssel verwendet.

Zu einem Array könnte man die Daten folgendermaßen zusammenfassen: Sie legen einen Array-Namen fest, beispielsweise `$kleidung`. Der Indexwert wird einer eckigen Klammer übergeben und dann weisen Sie der Array-Liste die Werte zu:

```
$kleidung[1] = "Hosen";
$kleidung[2] = "Röcke";
$kleidung[3] = "Kleider";
$kleidung[4] = "Jacken";
$kleidung[5] = "Pullover";
```

Eine andere Schreibweise – diese ist eher die klassische – zum Erstellen einer Array-Liste funktioniert auch. Die Zuweisung der Werte erfolgt über die Funktion `array()`. Das sieht dann so aus:

```
$kleidung = array (1=>"Hosen", 2=>"Röcke", etc.);
```

Dies erspart zwar die Wiederholung des Array-Namens, ist aber nach unserem Verständnis etwas weniger übersichtlich. Achten Sie gut auf die Syntax: Sie schreiben den Schlüssel für den Wert und dann durch einen Pfeil getrennt den Wert selbst.

Wenn Sie, nachdem Sie ein Array erstellt haben, den `echo`-Befehl verwenden und lediglich schreiben

```
echo $kleidung;
```

ist das Ergebnis nicht besonders spannend (Abbildung 3.19), denn Sie erhalten lediglich das Wort *Array* als Rückgabewert. Immerhin wissen Sie dadurch, dass das Erstellen des Arrays geklappt hat. Um die vollständige Array-Liste angezeigt zu bekommen, müssen Sie das Array durchlaufen, was weiter unten besprochen wird. Ansonsten können Sie auf einzelne Elemente eines Arrays zugreifen. Dies geschieht, indem Sie den entsprechenden Indexwert bzw. Schlüssel ansprechen, also beispielsweise schreiben:

```
echo $kleidung[1];
```

Sie müssen den Elementen keine Indexwerte zuweisen. PHP sorgt auch selbst für die Indizierung, wobei eine Besonderheit zu beachten ist: Die Indizierung von PHP beginnt bei Null und nicht bei Eins. Im Array

```
$kleidung[] = "Hosen";
$kleidung[] = "Röcke";
$kleidung[] = "Kleider";
```

wäre das nullte Element `Hosen`. Sie müssten sich also auf `$kleidung[0]` beziehen, wenn PHP auf das Element mit dem Wert `Hosen` zugreifen soll. Schreiben Sie ein Script wie in Abbildung 3.20, wird im Browser ausgegeben:

```
Wir haben schöne Kleider.
```

Wenn Sie das Array über die Funktion `array()` festlegen, schreiben Sie – sofern Sie PHP die Indizierung überlassen – einfach:

```
$kleidung = array("Hosen", "Röcke", "Kleider");
```

Wollen Sie auf `Kleider` zugreifen, brauchen Sie den Indexwert [2].

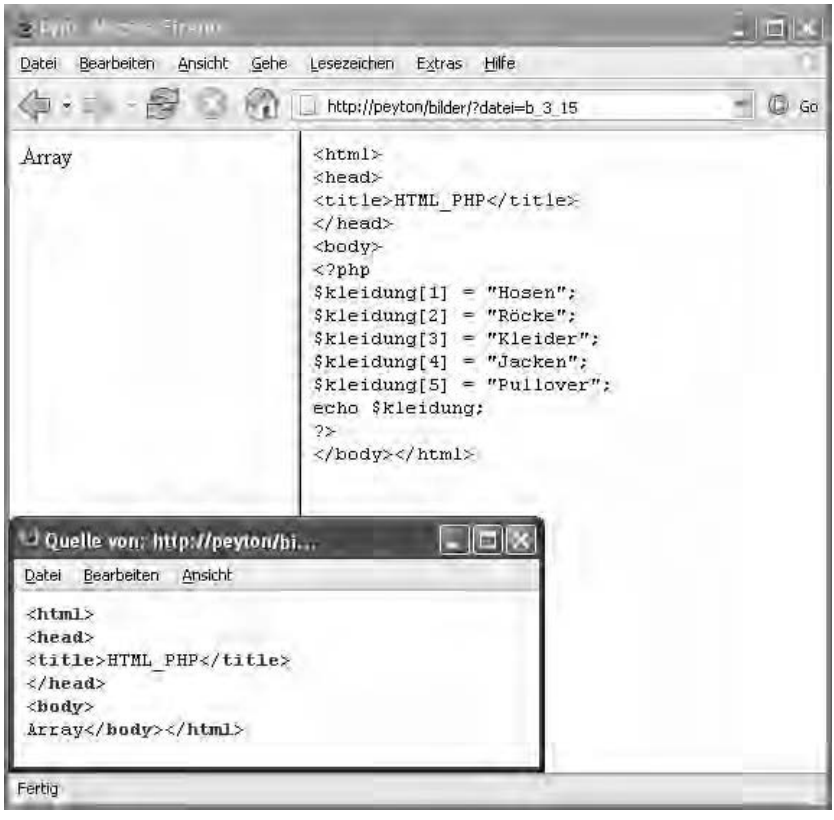

Bild 3.19: Eine Array-Liste erstellen – die Ausgabe mit `echo` im Browser

Assoziative Arrays

Der Indexwert muss keine Zahl sein, sondern kann auch eine Zeichenkette sein. Man spricht dann von assoziativen Arrays. Es kann auch vorkommen, dass Sie mitunter den Begriff Hash-Arrays lesen. So könnten Sie auch folgenden Code schreiben:

```
$kleidung['lieferant1'] = "Hosen";
$kleidung['lieferant2'] = "Röcke";
```

Dabei müssen Sie wieder daran denken, dass PHP auch bei den Schlüsseln, die Sie als String in die eckige Klammer schreiben, sensibel auf Groß- und Kleinschreibung reagiert. Die andere Schreibweise ist:

```
$kleidung = array ('lieferant1' =>"Hosen", 'lieferant2' =>"Röcke", ↵
'lieferant3' => "Kleider");
```

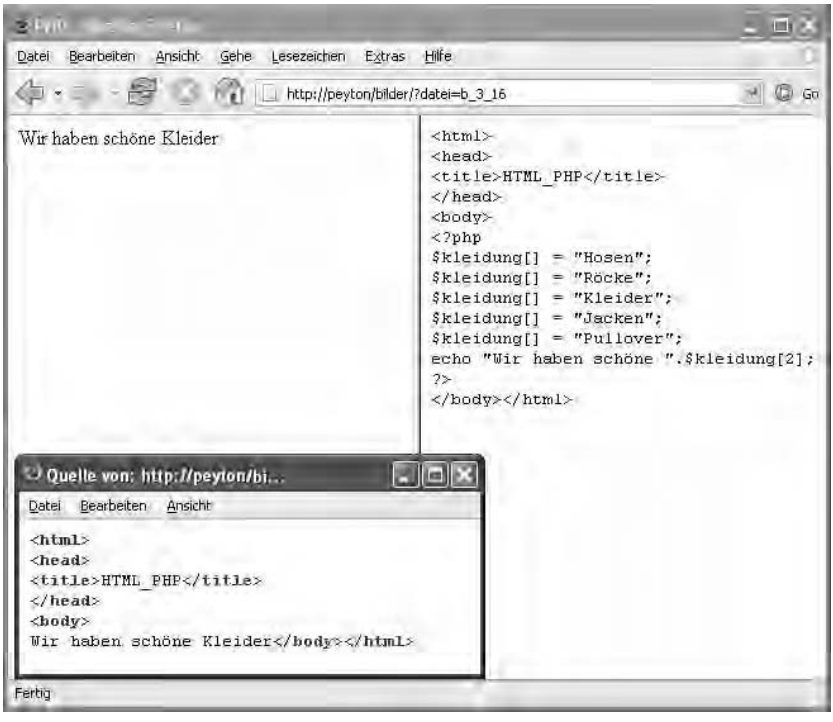

Bild 3.20: Auf Elemente eines Arrays zugreifen

Die Wörter (bzw. Strings) müssen nicht in Anführungszeichen gesetzt werden, wenn der Schlüssel nur ein Wort enthält. Es ist aber zu empfehlen, bei den Schlüsselnamen der assoziativen Arrays immer Anführungszeichen zu verwenden (eingebürgert haben sich für die Schlüssel einfache Anführungszeichen), da Sie PHP damit etwas Arbeit abnehmen und eine mögliche Fehlerquelle entschärfen. Wenn Sie keine Anführungszeichen verwenden, schaut PHP nach, ob es eine Konstante mit dem entsprechenden Namen gibt, und verwendet gegebenenfalls den Wert dieser Konstante. Setzen Sie Anführungszeichen, ersparen Sie PHP diesen Arbeitsschritt und laufen nicht Gefahr, einen zuvor als Konstante definierten Schlüssel einzusetzen. Schauen Sie sich das Beispiel an:

```
define("lieferant1", "irgendwas");
$kleidung[lieferant1] = "Hosen";
echo "<br>Test1 = ".$kleidung['lieferant1'];
$kleidung['lieferant1'] = "Hosen";
echo "<br>Test2 = ".$kleidung['lieferant1'];
```

Dieses kleine Script wird Folgendes ausgeben:

```
Test1 =
Test2 = Hosen
```

Das erste `echo` wird kein Array-Element mit dem Schlüssel `lieferant1` ausgeben, da als Schlüssel der Wert der Konstanten `lieferant1` – also `irgendwas` – verwendet wurde. Erst mit der nächsten Zuweisung wird, da der Schlüssel in Anführungszeichen gesetzt ist, das Array-Element mit dem Schlüssel `lieferant1` mit dem Wert `Hosen` belegt.

Arrays ergänzen

Arrays lassen sich auch mühelos ergänzen. Sie hängen der vorhandenen Liste einfach die gewünschten Elemente an. Wenn Sie die Elemente nicht spezifizieren, wird jedes Element mit der nächsten logischen Zahl indiziert. Ein neuer Wert in der Array-Liste `$kleidung`, beispielsweise `$kleidung[] = "Socken";`, wäre automatisch das dritte Element.

Der Vorteil von Arrays liegt neben der einfachen Eingabe (d.h. ein Variablenname statt ganz viele für jeden unterschiedlichen Wert) in der bequemen Bearbeitung der einzelnen Elemente. Dies werden wir an vielen Beispielen in den Kapiteln weiter hinten demonstrieren.

3.11.1 Der Umgang mit Arrays – Beispiele

In diesem Abschnitt machen wir Sie etwas weitergehend mit Arrays bekannt. Weitere Möglichkeiten der Analyse und Bearbeitung stellen wir dann im nächsten Kapitel vor, das sich speziell mit vordefinierten Funktionen, also auch mit einer Vielzahl von Array-Funktionen, befasst.

Probieren Sie am besten ein einfaches Script mit einem Array aus.

Öffnen Sie Ihren Editor und beginnen Sie eine neue Datei. Nach dem üblichen HTML-Header erstellen Sie eine Array-Liste. Sie können dazu das folgende kleine Script benutzen. Beachten Sie die beiden Zeilen zwischen den Zeichen `/*` und `*/`. Es handelt sich um einen Kommentar, der nicht ausgeführt wird. Er zeigt die alternativen Schreibweisen, das Array zu erstellen.

```
<html>
<head>
<title>comment</title>
</head> <body>
<?php
$Kurse = array('Montag'=>"Weben", 'Dienstag'=>"Stricken", ↵
'Mittwoch'=>"Tanzen", 'Donnerstag'=>"Malen");
/*
$Kurse['Montag']="Weben";
$Kurse['Dienstag']="Stricken";
*/
?>
</body> </html>
```

Um einen spezifischen Wert eines Arrays direkt anzusprechen, müssen Sie sich, wie oben bereits erwähnt, auf den Schlüssel beziehen. Schreiben Sie beispielsweise die folgende Zeile in das Script:

```
echo "<p>Neu ist unser Kurs in ".$Kurse['Montag']."</p>";
```

Wir wählen hier übrigens die »saubere« Schreibweise mit Verkettung (die Punkte, Sie verketten also den String, die Variable mit dem Array-Element und das </p>). Wir wollen Ihnen aber nicht verschweigen, dass auch die einfachere Zeile:

```
echo "<p>Neu ist unser Kurs in $Kurse[Montag] </p>";
```

funktionieren würde. Es gibt aber auch Fälle, in denen diese Schreibweise nicht das Gelbe vom Ei ist, und deswegen raten wir Ihnen dringend, sich an die Syntax mit Verkettung zu gewöhnen.

Speichern Sie das Dokument als PHP-Datei und testen Sie es in Ihrem Browser.

Die Existenz von Elementen prüfen

Mitunter kommt es vor, dass Sie prüfen möchten, ob ein bestimmtes Element in einem Array vorkommt. Statt das Array durchlaufen zu müssen, gibt es seit PHP4 die Funktion `in_array()`. Als Argumente schreiben Sie das zu suchende Element in die Klammer sowie das zu durchsuchende Array. Sie verwenden dabei einen einfachen `if`-Befehl (wir greifen hier vor; ausführlich wird die `if`-Anweisung erst in Kapitel 5, *Kontrollstrukturen*, erklärt). Das Ganze könnte – bezogen auf unser obiges Beispiel-Array `$kleidung` – so aussehen wie in Abbildung 3.21:

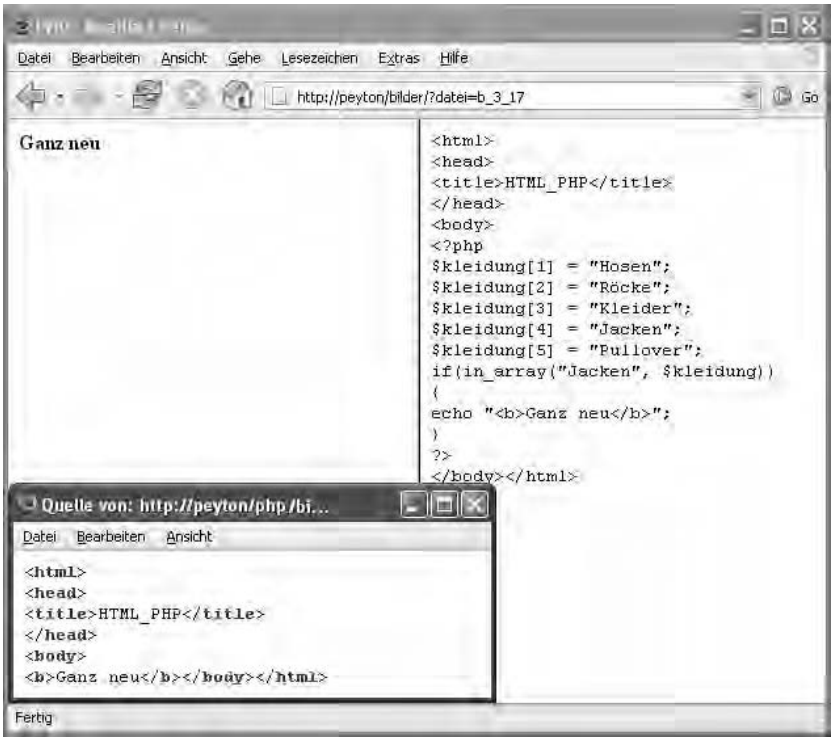

Bild 3.21: Mit `in_array` prüfen Sie, ob ein Element existiert

Eine Array-Liste durchlaufen

Eine Möglichkeit zum Durchlaufen eines Arrays bietet der Befehl `foreach()`. PHP liest das Array Element für Element und bei jedem Durchgang wird der Wert des aktuellen Elements des Arrays vorübergehend einer Variablen zugewiesen, und der interne Array-Zeiger um eins erhöht. Dadurch wird beim nächsten Durchgang automatisch das nächste Element ausgewertet. Die Anweisung `foreach` ist eine so genannte Schleife, die wir in Kapitel 5, *Kontrollstrukturen,* ausführlicher erklären. Die allgemeine Syntax ist die folgende:

```
foreach(arrayname as $temp)
{
Anweisung;
}
```

In der Variablen `$temp` – der Name kann beliebig gewählt werden – wird jedes Element temporär gespeichert. Der Name der Variablen wird nach `as` angegeben. Damit die Namen untereinander erscheinen, wird ein `<br>` (achten Sie auf die Anführungszeichen) in die Zeile geschrieben.

Daher schreiben Sie in das Script zum Durchlaufen eines Arrays:

```
foreach($Kurse as $ziel)
{
echo $ziel."<br>";
}
```

Die nächste Abbildung zeigt das Script zum Durchlaufen des Arrays `$Kurse`. Das Ergebnis sehen Sie im Browser. Achten Sie auch drauf, dass die Liste nett untereinander steht; das bewirkt natürlich das `<br>`.

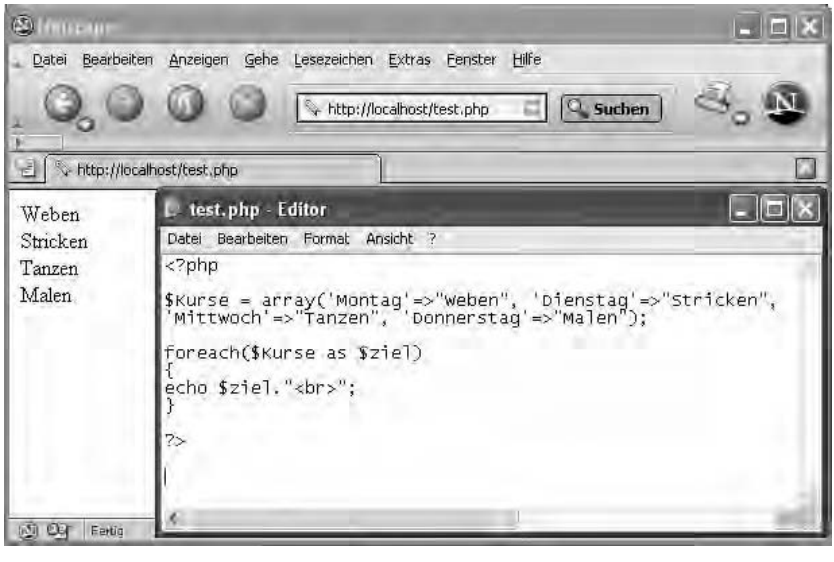

Bild 3.22: Eine Array-Liste durchlaufen

Assoziative Arrays durchlaufen

Betrachten wir auch noch die Möglichkeit, assoziative Arrays zu durchlaufen, also die, in denen die Schlüssel aus Zeichenketten bestehen. Nehmen wir eine neue Array-Liste an:

```
$reisen = array('Norden'=>"Finnland", 'Westen'=>"Irland", ↵
'Osten'=>"Polen");
```

Um die Schlüssel und Werte anzuzeigen, schreiben Sie:

```
foreach($reisen as $key=>$wert)
{
echo "$key = $wert<br>";
}
```

Die Variable `$key` enthält temporär jeden Schlüssel des Arrays und die Variablen `$wert` den zu diesem Element gehörenden Wert. Mit `foreach` wird das Array Element für Element durchlaufen.

Als Ergebnis erhalten Sie eine Ausgabe, die in Abbildung 3.23 zu sehen ist.

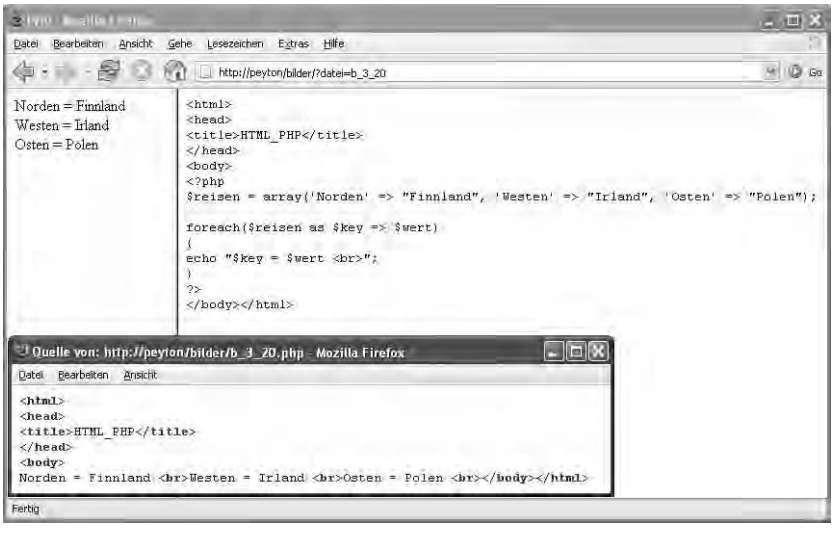

Bild 3.23: Die Ausgabe der Elemente eines assoziativen Arrays im Browser

3.11.2 Mehrdimensionale Arrays

Auch wenn mehrdimensionale Arrays nicht unbedingt zu den (notwendigsten!) Grundlagen gehören, wollen wir sie kurz ansprechen. Bei mehrdimensionalen Arrays nutzen Sie den `array`-Befehl innerhalb eines übergeordneten Arrays. Dadurch können Sie ein Array erstellen, das sogar noch mehr Informationen enthält als ein »normales« Array. Der Trick ist folgender: An Stelle von Strings oder Zahlen dienen andere Arrays als Werte. Wenn Sie also beispielsweise zwei Arrays haben mit den Namen

$inlandreisen und $auslandreisen, könnte die Syntax für das mehrdimensionale Array so aussehen wie in der dritten Zeile:

```
$inlandreisen=array("Hamburg", "Frankfurt", "Berlin");
$auslandreisen=array("Finnland", "Island", "Schweden");

$liste1=array('inland'=>$inlandreisen, 'ausland'=>$auslandreisen);
```

Um ein Element eines mehrdimensionalen Arrays anzusprechen, müssen Sie (sofern Sie selbst keine Schlüssel gesetzt haben) die Reihe mitzählen. Die Schreibweise ist folgende:

```
$liste1['inland'][1]
```

Dies wäre also im Beispiel Frankfurt, das zweite Element im Ursprungs-Array $inlandreisen.

3.12 Reguläre Ausdrücke

Wie eingangs bereits gesagt, sind reguläre Ausdrücke sehr nützlich und zeitsparend, vor allem im Zusammenhang mit Zeichenketten. Sie haben allerdings die merkwürdige Eigenart, im Prinzip leicht zu verstehen, aber in der Praxis des Programmierens etwas bis ziemlich verzwickt zu sein. Wir können in dieser Einführung nur die Grundlagen behandeln und Ihnen den Rat mit auf den Weg geben, etwas tiefer in die Einsatzmöglichkeiten der regulären Ausdrücke einzusteigen, sofern Sie zukünftig mit PHP programmieren möchten oder müssen.

Sie können sich reguläre Ausdrücke als ein System sich entsprechender Muster vorstellen. Im Prinzip leisten die Funktionen für reguläre Ausdrücke Folgendes: Man kann mit ihrer Hilfe vorher definierte Muster mit Text-(Teil-)Zeichenketten abgleichen, um entweder Übereinstimmungen zu finden oder eben nicht. Sofern gewünscht, können Sie Muster oder Zeichenketten bei gefundenen Übereinstimmungen zwischen dem Muster und der (Teil-) Zeichenkette dann auch ersetzen lassen. Das Verfahren ist folgendermaßen: Man schreibt zunächst das Muster, dann verwendet man eine der Funktionen von PHP und wendet das Muster auf einen Text (bzw. eine Zeichenkette) an.

Im Wesentlichen kennt PHP zwei Funktionen zum Vergleichen von Mustern und Funktionen zum Ersetzen von übereinstimmendem Text durch einen anderen Text.

Wir beginnen damit, ein Muster zu definieren und eine Übereinstimmung zu suchen.

3.12.1 Muster definieren

Bevor Sie die Funktionen der regulären Ausdrücke verwenden können, müssen Sie Muster definieren. Die einzusetzende Funktion benutzt dieses Muster dann für den Vergleich. Um Muster zu definieren, gebrauchen Sie bestimmte Symbole/Zeichen, die gruppiert (und u.U. zu Klassen zusammengesetzt) werden. Die Kombination von Symbolen, Gruppen (und Klassen) bildet das Muster.

Symbole/Zeichen

Der gebräuchlichste Typ bei der Definition von Mustern ist das einfache, wortwörtlich zu nehmende Zeichen. Wenn Sie beispielsweise als Muster *a* verwenden, wird auch nur mit dem Buchstaben *a* eine Übereinstimmung gefunden. Diese Definition reicht in manchen Situationen aus, aber leistungsstärker wird der Einsatz von regulären Ausdrücken, wenn Sie spezielle Symbole verwenden, die jeweils eine eigene Bedeutung jenseits ihrer »wortwörtlichen« haben. Der Unterschied zwischen den einfachen Zeichen (also *a* stimmt nur mit *a* überein) und den speziellen Symbolen (auch als Metazeichen bezeichnet) ist der, dass Metazeichen unter anderem weiter reichende Übereinstimmungen finden bzw. nach bestimmten Wiederholungen eines Zeichens suchen. Z.B. entspricht der Punkt (.), der ein solches Zeichen ist, einem beliebigen Zeichen mit Ausnahme des Zeilenwechsels (».« gleich a, b, c, 1, 3 oder ! etc.).

Die abgebildete Tabelle listet diese Zeichen und ihre Bedeutung auf (die Symbole mit der geschweiften Klammer werden auch als Grenzen bezeichnet, sie legen damit fest, wie oft ein Zeichen oder Zeichenbereich gesucht wird).

Zeichen	Übereinstimmung
.	jedes beliebige Zeichen
^a	stimmt überein mit einem String, der mit dem Zeichen beginnt, das nach dem »Dach« steht.
a$	stimmt überein mit einem String, der mit dem Zeichen endet, das vor dem Dollarzeichen steht.
a+	stimmt überein mit dem String, der vor dem + steht und der mehrfach, mindestens aber einmal auftritt.
a?	stimmt überein mit einem String, der höchstens einmal auftritt.
a*	stimmt überein mit einem String, der beliebig häufig auftreten kann – auch keinmal.
\n	stimmt mit einer neuen Zeile überein.
a{2}	stimmt überein mit einem String, der zweimal auftritt.
a{1,}	stimmt überein mit einem String, der mindestens einmal auftritt.
a{1,3}	stimmt überein mit einem Sting, der mindestens einmal und höchstens dreimal auftritt.
[0-9]	stimmt überein mit irgendeiner Ziffer zwischen 0 und 9.

Tabelle 3.6: Metazeichen der regulären Ausdrücke

Lassen Sie uns dies anhand einiger konkreter Beispiele verdeutlichen.

Das Muster *f+* findet seine Entsprechung mit allen Zeichenbereichen, die mindestens ein *f* (oder aber mehr) enthalten, beispielsweise: *f+*, als Übereinstimmung gefunden werden *fisch*, *affe*, *schifffahrt*, *ffff* etc.

Das Muster *b{2,3}* passt zu einer Zeichenkette, die *b* mindestens zweimal, höchstens dreimal enthält. Es werden also gefunden: *ebbe* oder *blubbbad*, aber nicht *abc*.

Die Kombination des Punktes . (steht dafür, dass ein beliebiges Zeichen vorausgesetzt wird) mit dem Zeichen + ist ein Muster für ein beliebiges Zeichen, und davon muss es mindestens eines geben. Insofern wird .+ häufig eingesetzt, denn es entspricht als Muster (mindestens) einem Zeichen. Findet sich ein Match, würde die Prüfung also *true* zurückgeben.

Gruppen

Eine Gruppe besteht aus den Zeichen, wenn sie zusammengefasst in Klammern geschrieben werden. Eine einfache Gruppe wäre *(abc)*, wobei eine Gruppierung hier überflüssig ist, da *abc* das gleiche Resultat erzeugt: Es passt zur Zeichenkette *abc*. Interessanter ist die Gruppierung in Kombination mit der geschweiften Klammer. Schauen Sie noch einmal in die obige Tabelle: *a{2}* passt zu *aa*, d.h. der Buchstabe *a* tritt zweimal auf. Wenn Sie *(abc){2}* schreiben, findet sich bei *abcabc* eine Übereinstimmung. Deutlicher wird der Unterschied bei folgendem Beispiel eines gruppierten Musters:

Das Muster *schiff+* findet eine Übereinstimmung, wenn *schif* gefolgt wird von (mindestens) einem weiteren *f*. *(schiff)+* hingegen entspricht nur einer Zeichenkette, die mindestens einmal *schiff* oder mehrmals *schiff* hintereinander enthält. Alles klar?

3.12.2 Funktionen der regulären Ausdrücke

Zwei vordefinierte Funktionen suchen Übereinstimmungen mit festgelegten Mustern. Die eine ist `ereg()` und die andere `eregi()`. Der Unterschied zwischen beiden ist ganz einfach: `ereg` unterscheidet zwischen Groß- und Kleinschreibung und `eregi` nicht.

Beide Funktionen geben *true* zurück, wenn eine Übereinstimmung zwischen Muster und Zeichenkette gefunden wird, und ansonsten *false*. Ein Einsatz der Funktion(en) kann folgendermaßen aussehen:

```
ereg("definiertes Muster", "zu vergleichende zeichenkette");
```

Geschickter ist es, das Muster einer Variablen zuzuweisen:

```
$muster = "definiertes Muster";
$string = "zu vergleichende Zeichenkette";
echo ereg($muster, $string);
```

Vergleichen und Ersetzen

Während die Funktionen `ereg()` und `eregi()` beispielsweise sehr wirksam einsetzbar sind, um die Gültigkeit bestimmter Eingaben (z.B. in einem Formularfeld) zu testen, ist es in anderen Fällen wünschenswert, Eingaben zu verwandeln. Dann kommen die Funktionen

`ereg_replace()` oder `eregi_replace()`

ins Spiel.

Wie eingangs bereits erwähnt, demonstrieren wir den konkreten Einsatz regulärer Ausdrücke erst in einem späteren Kapitel des Buches, da wir Sie an dieser Stelle damit vermutlich überfordern würden. In Kapitel 23 *Tipps und Tricks* am Ende des Buches finden Sie dann zwei Beispiele für die Verwendung regulärer Ausdrücke.