Christine Peyton, André Möller

PHP 5 & MySQL 4

Der leichte Einstieg

Markt+Technik Verlag

3
PHP – Die Grundlagen

Eine neue Programmiersprache zu lernen muss damit beginnen, zunächst die Grundlagen der Sprache zu verstehen. Diese Grundlagen sollen in diesem Kapitel geklärt werden. Sie erfahren zunächst, wie ein PHP-Script aufgebaut wird und wie Sie HTML und PHP kombinieren. Dann werden beispielhaft wichtige und gebräuchliche Funktionen/Befehle vorgestellt und an kleinen Beispielen wird kurz erläutert, wie Sie Befehle benutzen. Ein wichtiger Baustein von PHP (und anderen Programmiersprachen) sind Variablen. Wir klären also, was Variablen sind, welche unterschiedlichen Variablentypen es gibt und wie Sie sie einsetzen.

Entscheidend für dynamische Webseiten ist Flexibilität. Damit ist gemeint, dass Scripts Bedingungen auswerten können und je nach dem Ergebnis eines Vergleiches (true/false) ein anderes Ereignis eintritt. Um diese Möglichkeit von Fallunterscheidungen nutzen zu können, müssen Sie Verzweigungen oder Kontrollstrukturen kennen lernen.

Keine Programmiersprache ohne Schleifen! Mit Schleifen geben Sie an, wie oft ein bestimmtes Programmstück ausgeführt wird – wie Sie sich denken können, ein entscheidendes Element eines Scripts. Wir werden also, jeweils untermauert mit einigen Beispielen, erklären, wie Sie mit der `while`-Anweisung und der `for`-Schleife umgehen.

Im letzten 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 ihren Gebrauch an einem Beispiel demonstrieren.

Dieses Beispiel 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 die Grundlagen ab, behandelt aber bei weitem nicht alle Befehle und sonstigen Möglichkeiten von PHP. Dies entspricht dem im Vorwort bereits geschilderten projektorientierten Konzept dieses Buches, das versucht, Ihnen PHP an praxisnahen Beispielen unter dem

Einsatz von Befehlen, die tatsächlich gebraucht werden, nahe zu bringen. Falls Sie hier dennoch eine Erklärung schmerzlich vermissen: Werfen Sie einen Blick in die Befehlsreferenz, dort finden Sie unter Umständen das Gesuchte.

Die Scripts testen

Während Sie beim Erstellen von HTML-Seiten diese direkt im Browser betrachten können, benötigen Sie zum Testen von PHP-Dateien einen Server, der PHP unterstützt. Arbeiten Sie an Ihrem heimischen Computer und nicht am Server, müssen Sie die Scriptdateien also unter Umständen auf den Server hochladen. Sie benutzen dazu irgendein Standard-FTP-Programm, z.B. WS_FTP.

> **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 jedem Benutzer Zugang gewähren, um beispielsweise kostenlose Software herunterzuladen oder Dokumente zu beziehen.

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.

Das Grundgerüst eines PHP-Scripts

Sie brauchen zum Schreiben eines PHP-Scripts, wie Sie es von HTML gewohnt sind, einen Texteditor. Ein reines PHP-Script ohne HTML-Elemente sieht dann in seiner Struktur zunächst ganz einfach aus: Die ausführbaren PHP-Codes werden in die folgenden Anfangs- und Endezeichen 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 `%>`. Manche Programme, beispielsweise Dreamweaver, funktionieren mit den ASP-Tags besser als mit den PHP-Tags.

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 unbedingt erforderlich ist). Ein Semikolon nach einem Code zu vergessen, ist eine oft vorkommende Fehlerquelle, versuchen Sie also immer daran zu denken. 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>
```

Listing 3.1: Das Grundgerüst für ein Script mit HTML-Elementen und PHP-Code

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 Endezeichen `?>` 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 (Cookies setzen, Header-Informationen angeben) finden, 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.

Speichern

HTML-Seiten, die PHP-Elemente enthalten bzw. reine PHP-Scripts sind, 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-Scripte 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. Beachten Sie, dass Sie die Datei mithilfe des Webservers aufrufen, damit das enthaltene PHP-Script ausgeführt werden kann. Geben Sie dazu 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 dann lediglich: 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 die Auswertung übernimmt, verwiesen. Diese Methode hat allerdings viele Nachteile bei der Verarbeitung. 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 für die Lösung der ersten konkreten Aufgabe, die in ***Kapitel 4, Ein Kontaktformular erstellen***, beschrieben wird, zunächst zwei Dokumente erstellen, anstatt nur eine einzige PHP-Datei, die den PHP-Programmcode und HTML kombiniert.

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 im Browser aus. 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>
<?php
echo "hier steht ein neuer PHP-Code";
?>
</body></html>
```

Listing 3.2: Ein Script, in dem PHP und HTML gemischt werden

Bei dieser Methode beenden Sie also PHP, schreiben den HTML-Teil mit den jeweils benötigten Tags und beginnen PHP erneut.

Es ist aber auch möglich, HTML mithilfe 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 folgende Seite aus:

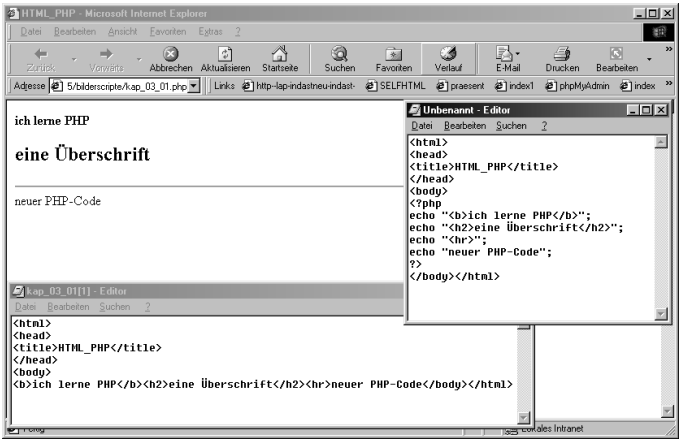

Bild 3.1: Die Ausgabe im Browser – ein fett formatierter Satz (`<b>`) *und eine horizontale Linie* `<hr>`

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, als ob es in einer Zeile stehen würde:

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

Im Browser wird als Ausgabe auch 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 mitunter übersichtlicher machen. Auch per Tabulator eingerückte Teile erhöhen manchmal 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!

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. Sie brauchen ein ganz einfaches »Script«:

```
<?php
echo phpinfo();
?>
```

Der Browser zeigt daraufhin die erwähnte Tabelle mit den Informationen über PHP. Riskieren Sie ruhig einen Blick.

Bild 3.2 gibt einen Ausschnitt der Tabelle wieder.

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"'`;

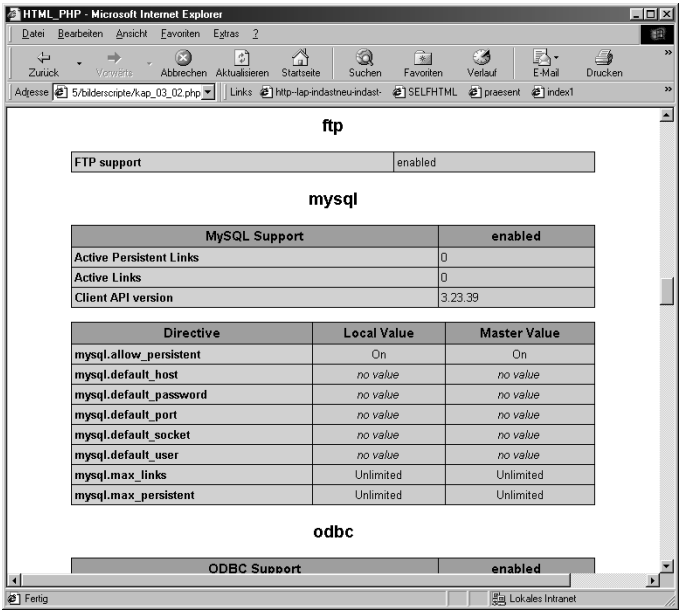

Bild 3.2: Mit `phpinfo()` *können Sie sich Informationen über die spezifische PHP-Installation ausgeben lassen.*

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 `<font color="red">`. Diese Anführungszeichen müssen natürlich auch maskiert werden, wenn sie Teil einer PHP-Anweisung sind. Das sieht dann so aus:

```
echo "<font color=\"red\">";
```

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

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

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

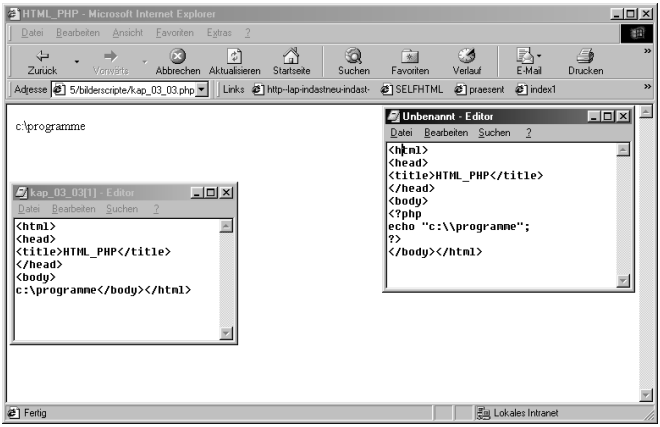

Bild 3.3: PHP erwartet einen zweiten Backslash.

Auch für Zeilenumbrüche in PHP (nicht zu verwechseln mit dem `<br>`-HTML-Tag) gibt es ein Zeichen mit einem Backslash: `\n`.

Die folgende Tabelle gibt einen Überblick über die Zeichen, die Sie verwenden können, damit bestimmte Zeichen ausgegeben werden.

Zeichenfolge	Effekt
\n	neue Zeile
\r	Wagenrücklauf
\t	horizontaler Tabulator
\\	Backslash
\$	Dollarsymbol
*	doppelte Anführungszeichen

Tabelle 3.1: Ein Überblick über die Zeichen, die zur Ausgabe spezieller Zeichen verwendet werden

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 eintritt, 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 oft 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 eingegebenen Wert
```

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:

```
<?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!

Tipp

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

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«.

Syntax und Wertezuweisung

Der Name einer Variablen ist frei wählbar. Die Schreibkonvention erlaubt Buchstaben, Zahlen und Unterstriche. Der Name darf keine Leerstellen oder nicht alphanumerische 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 Text 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 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 = "2001";
```

Im Regelfall werden Variablen jeweils mit dem neuen, in der Reihenfolge des Scripts zuletzt zugewiesenen Wert überschrieben (changed on the fly).

```
$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 Bild 3.4.

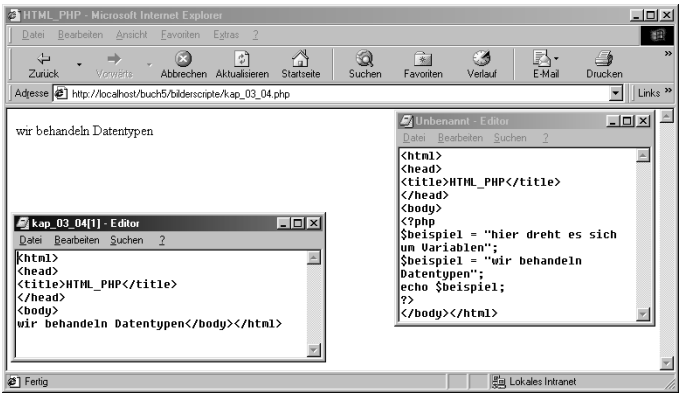

Bild 3.4: Variablen werden überschrieben.

Variablennamen unterscheiden zwischen Groß- und Kleinschreibung (sie sind *case-sensitiv*). `$Nummer` und `$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 also vor kryptischen Abkürzungen.

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 $stadt;
echo "$code $stadt";
```

Bild 3.5 zeigt den Code und die Ausgabe.

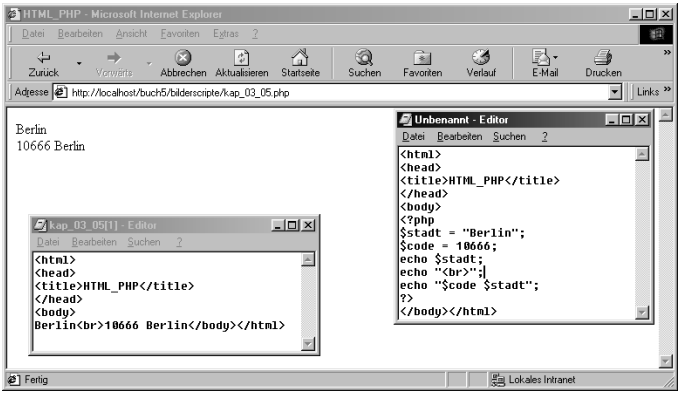

Bild 3.5: Die Inhalte von Variablen ausgeben

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, behält sie diesen 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;
```

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`

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

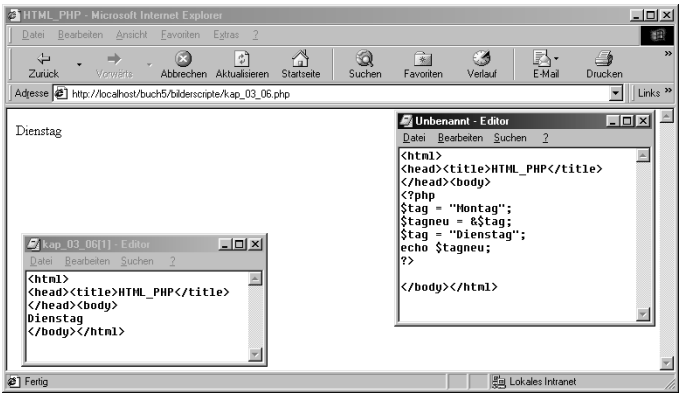

Bild 3.6: Zuweisung von Variablen per Referenzierung

Variablen erweitern/verketten

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:

Hätten Sie 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";
```

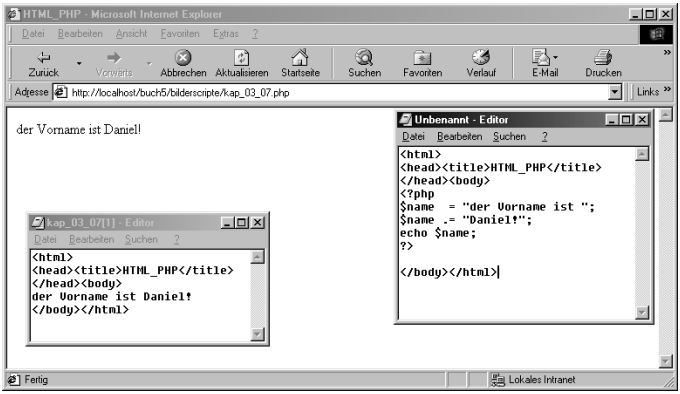

Bild 3.7: Die Ausgabe im Browser

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 Bild 3.8 sehen Sie den Code und die Ausgabe.

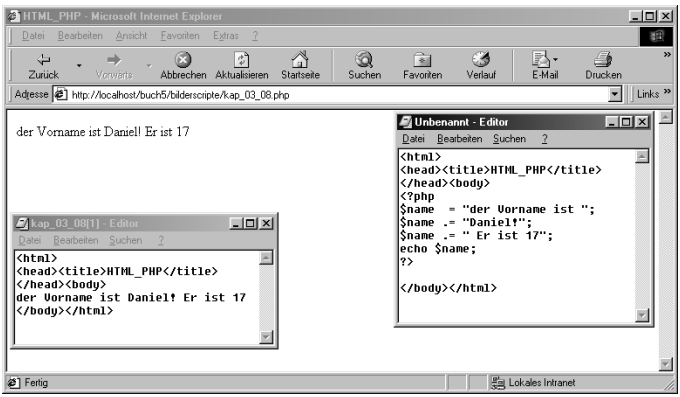

Bild 3.8: Variablen können erweitert werden

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*.

Bild 3.9 zeigt den Code und die Ausgabe im Browser.

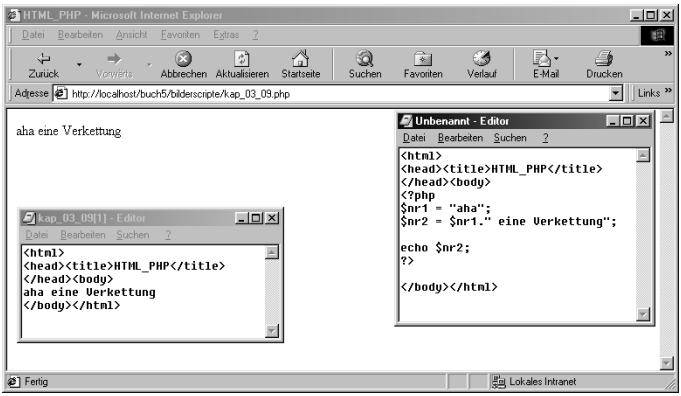

Bild 3.9: Variablen und Text können verkettet werden.

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 Bild 3.10. Durch eine weitere Verkettung wird der Variablen `$nr4` ihr Wert zugewiesen.

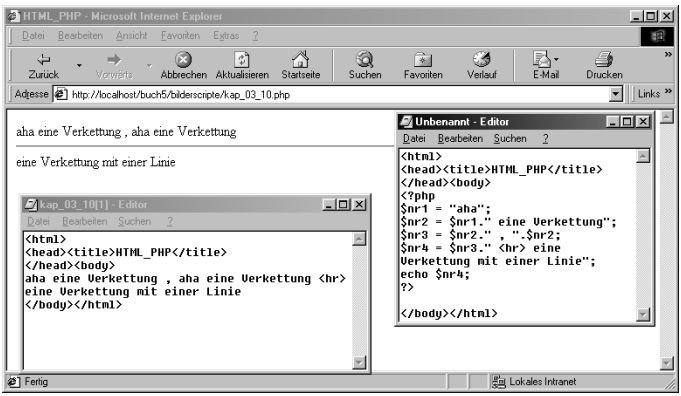

Bild 3.10: Noch mehr Elemente in der Verkettung

Hinweis

Statt die Werte der Variablen im Script festzulegen, können sie auch übergeben werden, beispielsweise durch User-Eingaben in ein Formular. 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 und $HTTP_POST_VARS` verwendet.

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 schon 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 *wahr* zurück, wenn die Variable existiert. 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` ist oder eine leere Zeichenkette.

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;
```

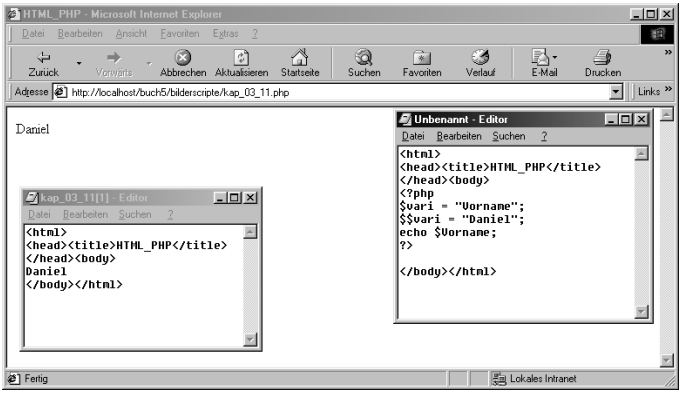

Bild 3.11: Die Wirkungsweise dynamischer Variablen

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 behandelt den Wert entsprechend.

Als Grobunterscheidung kann man die Datentypen zunächst unterteilen in:

- Zeichenketten, auch als String bezeichnet
- Zahlen
- Arrays

Genau genommen gibt es auch noch die Typen Boolean (kennt nur *true* oder *false*) und Object.

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"
```

Im Zusammenhang mit dem Befehl `echo` spielt es übrigens eine Rolle, ob Sie für die Ausgabe doppelte oder einfache Anführungszeichen gebrauchen. Der Unterschied ist der, dass bei doppelten Anführungszeichen die Variablen mit ausgewertet werden. Würden Sie stattdessen `echo 'Hallo, $vorname';` schreiben, also einfache Anführungszeichen benutzen, würde `'Hallo, $vorname'` ausgegeben werden. Das ist meistens nicht Sinn der Sache.

Zeichenketten lassen sich in vieler Hinsicht manipulieren und bearbeiten. Ein paar Funktionen wurden bereits weiter oben erwähnt, im Lauf der nächsten Kapitel werden Sie weitere kennen lernen.

Zahlen

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

Als Integer können beispielsweise gelten:

- 10
- -10

Beispielwerte für den Typ Double wären:

- 10.50;
- -10.50;

Werte inkrementieren

Im Verlauf der nächsten Kapitel werden wir demonstrieren, wie Sie Zahlen manipulieren und ihnen z. B. ein bestimmtes Format zuweisen können. An dieser Stelle beschreiben wir noch, wie Sie den Wert einer Variablen jeweils um eins hochzählen, ein Vorgang, der als Inkrementieren bezeichnet wird und in einem Script sehr häufig vorkommt. Denkbar ist das folgende Vorgehen:

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

Gefälliger 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:

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

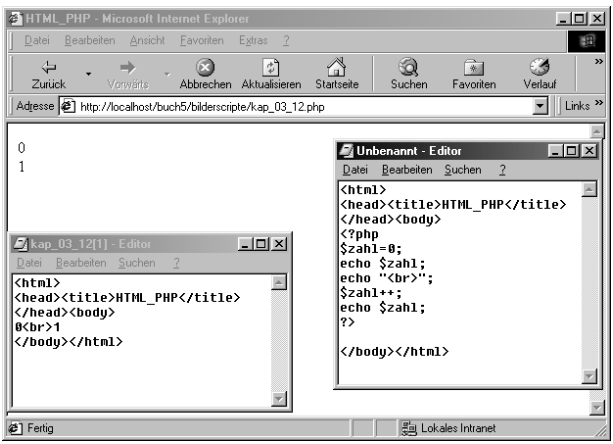

Bild 3.12: Zahlen mit ++ inkrementieren

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

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

Arithmetische Operatoren

Im Zusammenhang mit dem Datentyp Zahlen werden Sie in einem Script 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 3.2: 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.

Bild 3.13 zeigt beides, das Script im Editor und die Ausgabe im Browser.

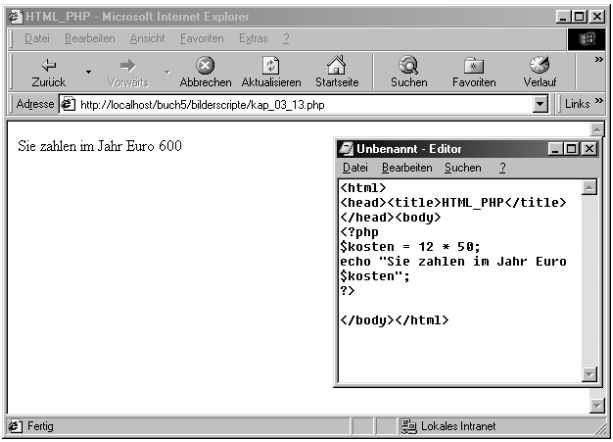

Bild 3.13: Für Berechnungen setzen Sie die üblichen Operatoren ein.

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;
```

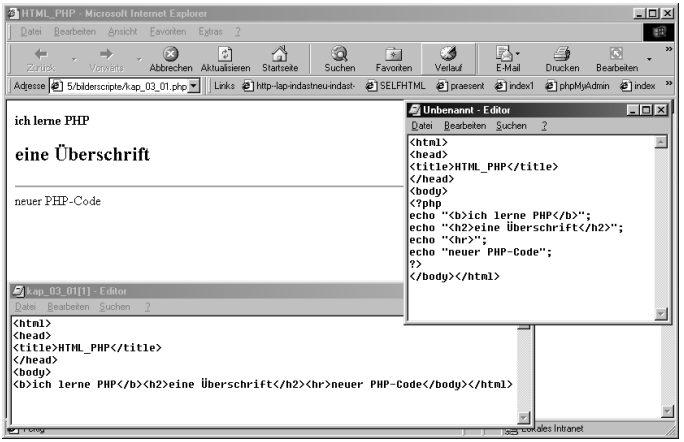

Bild 3.14: Eine einfache Berechnung – definieren Sie Variablen, um ein Script flexibel zu halten.

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

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 gerechnet 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)
```

Datentypen feststellen und festlegen

Um sicher zu wissen, 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);
?>
```

Listing 3.3: Ein kleines Script zur Rückgabe von Datentypen im Browser

Der Browser zeigt damit untereinander die Datentypen (wegen
) Integer und String.

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 Zahl. 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.

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 Schreibkonvention 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

Tabelle 3.3: Arrays sind strukturiert wie Tabellen. Im Beispiel werden Zahlen als Schlüssel verwendet.

Index oder Schlüssel	Wert
4	Jacken
5	Pullover

Tabelle 3.3: Arrays sind strukturiert wie Tabellen. Im Beispiel werden Zahlen als Schlüssel verwendet. (Forts.)

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";
```

Listing 3.4: So erstellen Sie ein Array.

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

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

Dies erspart zwar die Wiederholung des Wortes *array*, ist aber nach unserem Verständnis weniger übersichtlich. Achten Sie 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 (Bild 3.15), 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. Dies wird weiter unten besprochen. 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];
```

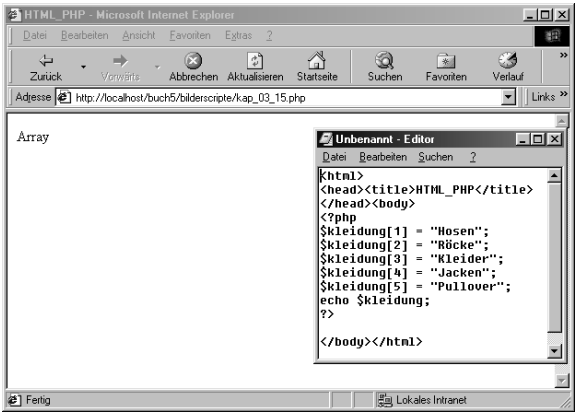

Bild 3.15: Eine Array-Liste erstellen – die Ausgabe mit echo im Browser

Sie müssen den Elementen keine Indexwerte zuweisen. PHP sorgt auch selbst für die Indizierung, wobei eine Besonderheit zu beachten ist: Diese Indizierung 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 Bild 3.16, wird im Browser ausgegeben: *wir haben schöne Kleider*.

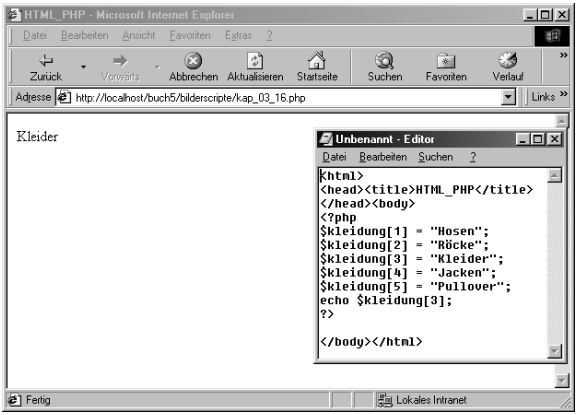

Bild 3.16: Auf Elemente eines Arrays zugreifen

Wenn Sie das Array über `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].

Assoziative Arrays

Der Indexwert muss keine Zahl, 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 Worte (bzw. Strings) müssen nicht in Anführungszeichen gesetzt werden, wenn der Schlüssel nur ein Wort enthält. Die andere Schreibweise ist:

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

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 relativ einfachen Beispielen in den Kapiteln weiter hinten demonstrieren.

Der Umgang mit Arrays – Beispiele

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 Listing 3.5 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>
```

Listing 3.5: zeigt ein Script mit einer Array-Liste, erstellt mit der Funktion `array()`.

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). 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 ist es eigentlich ratsam, sich an die Syntax mit Verkettung zu gewöhnen.

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

Arrays auswerten

PHP bietet eine Reihe von Möglichkeiten, eine Array-Liste auszuwerten. Wie Sie auf eines der Elemente zugreifen, haben wir bereits besprochen. Sie können aber beispielsweise auch die Anzahl der Elemente in einem Array feststellen. Das geht recht einfach. Der Befehl, den Sie dafür einsetzen, ist die Funktion `count()`.

Schreiben Sie unter die letzte Zeile des Arrays:

```
$allekurse = count($Kurse);
```

Damit legen Sie eine neue Variable fest (`$allekurse`), die die Anzahl der Elemente ausgeben wird.

Dann folgt der `echo`-Befehl. Sie benutzen also für die Ausgabe die eben definierte Variable, denn dieser wurde der Wert der gezählten Elemente zugewiesen:

```
echo $allekurse;
```

Betrachten Sie das Ergebnis im Browser. Sie müssten die Anzahl der Kurse angezeigt bekommen.

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 und das zu durchsuchende Array. Sie verwenden dabei einen einfachen `if`-Befehl. Das Ganze könnte – bezogen auf unser obiges Beispiel-Array `$kleidung` – so aussehen wie in Bild 3.17:

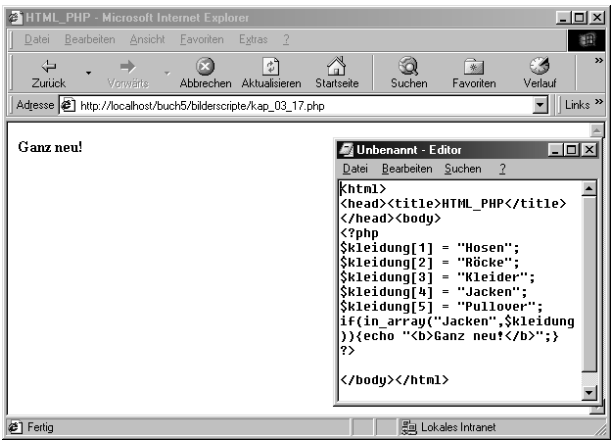

Bild 3.17: Mit `in_array` *prüfen Sie, ob ein Element existiert.*

Ein neues Array aus zwei Arrays bilden

Die Funktion `merge()` bildet aus zwei Arrays ein neues Array. *Merge* heißt übrigens so viel wie »verschmelzen«, die Bezeichnung passt also recht gut. Nehmen Sie an, Sie haben die folgende numerisch indizierte Array-Liste in einem Script:

```
$reisen[] = "Holland";
$reisen[] = "Schweiz";
$reisen[] = "Frankreich";
```

Außerdem gibt es eine zweite Liste:

```
$reisen1[] = "Schweden";
$reisen1[] = "Daenemark";
$reisen1[] = "Finnland";
```

Aus diesen beiden Arrays können Sie eine neue Array-Liste bilden. Dazu geben Sie einen beliebigen Array-Namen ein, als Argumente schreiben Sie den Namen des ersten Arrays und den Namen des zweiten Arrays in die Klammer:

```
$reisenneu = array_merge($reisen, $reisen1);
```

Um zu sehen, ob das funktioniert hat, lassen Sie die Elemente zählen und den Rückgabewert ausgeben. Ergänzen Sie also das Script um die folgenden Zeilen:

```
$allereisen = count($reisenneu);
echo $allereisen." Reisen im Angebot <br>";
```

Der Rückgabewert müsste in dem Fall *6* ergeben, ergänzt durch die Worte *Reisen im Angebot*. Achten Sie wieder auf die Leerstelle vor *Reisen*.

Arrays sortieren

Da Arrays ja Listen mit mehreren Elementen sind, eignen sie sich auch hervorragend zum Sortieren. Bestehen die Elemente aus Strings, kann man alphabetisch sortieren, bestehen sie aus Zahlen, numerisch. Überdies lassen sich entweder die Elemente (Werte) oder die Schlüssel in eine Reihenfolge bringen.

Die Funktion zum Sortieren der Elemente lautet `sort()` oder `rsort()`.

Die zweite Funktion erzeugt eine Sortierung von Z nach A.

Um die Werte der Array-Liste `$reisenneu` (also die Länder) zu alphabetisieren, ergänzen Sie das Script um die folgende Zeile:

```
sort($reisenneu);
```

Das nützt natürlich noch wenig, so lange Sie das Ergebnis nicht im Browser betrachten können. Folglich müssen Sie auf alle Elemente der Liste zugreifen, indem Sie die Liste durchlaufen.

Eine Array-Liste durchlaufen

Eine Möglichkeit zum Durchlaufen eines Arrays bietet der Befehl `foreach`, mit dem jeder Wert des Arrays vorübergehend einer Variablen zugeordnet wird.

Die Anweisung `foreach` ist eine so genannte Schleife (die wir im nächsten Abschnitt ausführlicher erklären). Die allgemeine Syntax ist die Folgende:

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

In der Variablen `$temp` wird jedes Element temporär gespeichert. Der Name der Variablen wird durch den Befehl `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:

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

Speichern Sie das Script ab und betrachten Sie das Ergebnis im Browser. Sie müssten nun eine alphabetisierte Liste der Länder erhalten. Das komplette Script dazu sieht aus wie in Bild 3.19.

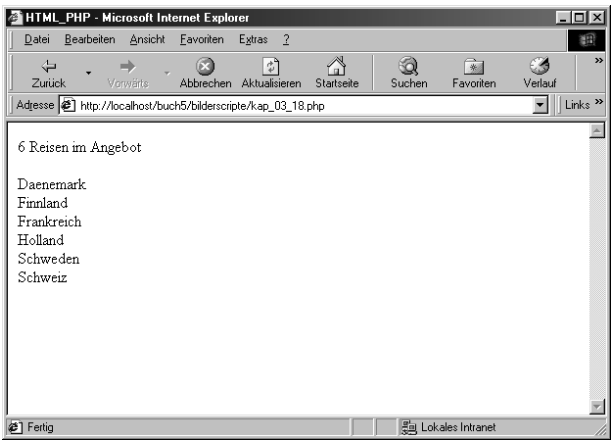

Bild 3.18: Eine alphabetisierte Liste wird ausgegeben.

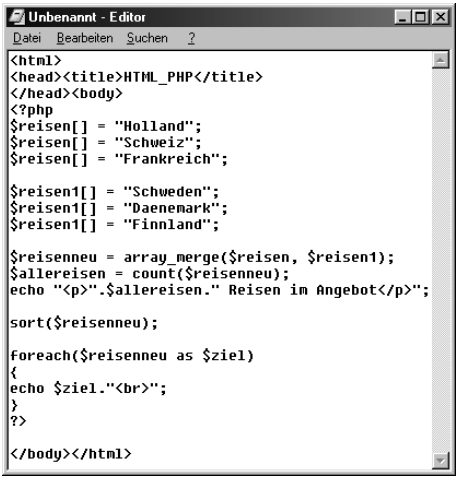

Bild 3.19: Das Script zum Verbinden von zwei Array-Listen, zum Sortieren der Elemente, zum Zählen der Elemente und zur Ausgabe der Liste

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` jeden Wert. Mit `foreach` wird das Array Element für Element durchlaufen.

Als Ergebnis erhalten Sie eine Ausgabe, die in Bild 3.20 zu sehen ist.

Varianten der sort-Funktionen

Mit der Funktion `sort()` werden die Elemente zusammen mit den Schlüsseln neu sortiert. Sollen trotz der Neusortierung die Schlüssel beibehalten werden, würden Sie stattdessen die Funktion `asort()` benutzen. Um nach den Indexwerten zu sortieren und zwar so, dass Sie die entsprechenden Werte behalten, set-

zen Sie die Funktion `ksort()` ein. Die Funktion `krsort()` sortiert auf Basis der Schlüssel von hinten nach vorn. Erwähnen wir zu guter Letzt die Funktion `shuffle()`. Damit werden die Elemente einer Array-Liste nach einem Zufallsprinzip in eine neue Reihenfolge gebracht.

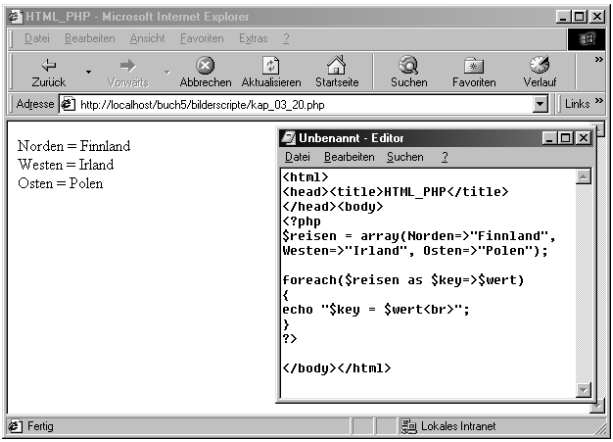

Bild 3.20: Die Ausgabe der Elemente eines assoziativen Arrays im Browser

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", "Franfurt", "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`.

Vordefinierte Funktionen

PHP kennt eine Fülle von vordefinierten Funktionen (im Gegensatz zu selbst definierten, die wir an dieser Stelle noch nicht erklären möchten). Durch Funktionen werden bestimmte Anweisungen, die PHP ausführen soll, übersichtlich zu Blöcken zusammengefasst.

Auch der eben eingesetzte Befehl `echo` ist eine Art Funktion, er bildet aber hinsichtlich der Schreibweise eine Ausnahme. Funktionen folgen normalerweise der Syntax:

```
Funktionsbezeichnung()
```

Sie brauchen also eine Klammer; diese Klammern nehmen (im Regelfall) ein oder mehrere Argumente oder Parameter auf. Sind es mehrere, werden die Argumente, also die Daten, die der Funktion übergeben werden, durch Kommata abgetrennt. Nur bei dem Befehl `echo` reichen Anführungszeichen, die Klammer ist optional.

Da Funktionen für die unterschiedlichsten Zwecke eingesetzt werden, werden sie systematisch in Kategorien eingeteilt. Dies erleichtert den Umgang mit ihnen, da es vor allem dem Einsteiger unmöglich ist, alle Funktionen auf Anhieb parat zu haben. Sind Sie also auf der Suche nach einer Funktion, die eine bestimmte Aufgabe durchführen soll, können Sie in Referenzbüchern oder Manuals in der entsprechenden Kategorie nachschlagen und müssen nicht eine unsortierte Liste von Hunderten Funktionen durchsehen.

Mit Funktionen aus folgenden Kategorien werden Sie zunächst (vermutlich) am häufigsten konfrontiert:

- Dateisystem-Funktionen
- Datums- und Zeitfunktionen
- Mail-Funktionen
- Mathematische Funktionen
- MySQL-Funktionen
- String-Funktionen

Es gibt eine Menge weiterer Kategorien; wir greifen hier – wie gesagt – die heraus, die besonders gängige und häufig eingesetzte Funktionen enthalten. In den folgenden Abschnitten stellen wir kurz aus den erwähnten Kategorien einige Funktionen vor.

Zeichenketten(String)-Funktionen

Eine String-Funktion (genau genommen ist dies keine Funktion, sondern ein Sprachkonstrukt) haben Sie bereits kennen gelernt:

`echo`

Mit `echo` werden alle Zeichenketten ausgegeben (Sie erfahren mehr über Zeichenketten in Abschnitt ***Datentypen***). Statt `echo` können Sie wahlweise auch

`print`

benutzen. Das ist reine Geschmackssache.

`trim()`

Der Befehl entfernt überflüssige Zeichen am Anfang und Ende einer Zeichenkette (Achtung, nicht in der Mitte!). Dieser Befehl wird häufig eingesetzt, wenn die Daten, die von PHP verarbeitet wurden, von einem Formular übertragen wurden. Es passiert Usern – bewusst oder unbewusst – recht häufig, dass bei der Eingabe von Daten überflüssige Leerstellen hinzugefügt werden. Als weitsichtiger Programmierer sollten Sie im PHP-Code dafür sorgen, dass diese Leerstellen entfernt werden. So kann der Code aussehen, wenn es beispielsweise ein Formularfeld `$nachname` gibt, das die Quelle der Daten ist:

```
$nachname =trim($nachname);
```

Eine Variante ist:

```
ltrim()
```

mit der die Leerstellen am Anfang einer Zeichenkette entfernt werden.

`strlen()`

Gibt die Länge einer Zeichenkette aus. Schreiben Sie zum Beispiel:

```
<?php
$testtext = "Was haben wir denn da?";
$length = strlen($testtext);
echo $length;
?>
```

Bild 3.21 zeigt den Code und die Ausgabe im Browser.

Mail-Funktionen

`mail()`

Mit dieser Funktion können Sie eine E-Mail im Text- oder HTML-Format an eine oder auch mehrere Empfänger verschicken. In der Klammer werden als Argumente erwartet: Empfänger, der Betreff und die Message.

```
mail("an@t-online.de", "testmail", "HalliHallo");
```

Um beim Verschicken einer Mail einen vernünftigen Absender zu erhalten, ist es empfehlenswert, auch einen optionalen Parameter, mit dem zusätzliche Header-Informationen angegeben werden, zu verwenden.

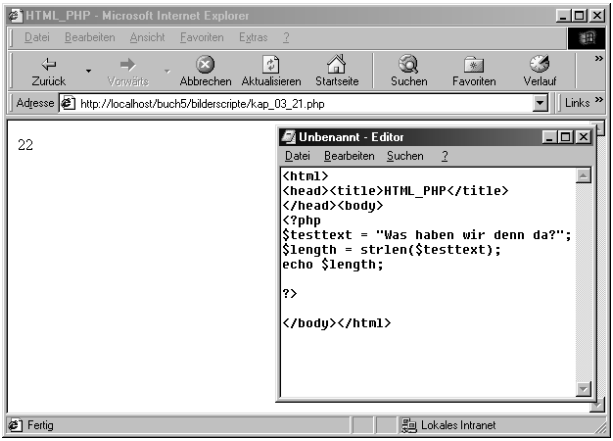

Bild 3.21: Mit `strlen()` *wird die Länge eines Strings ausgegeben.*

Hinweis

Den Header einer E-Mail-Nachricht sehen Sie in einem normalen E-Mail-Programm nicht. Mit diesem Header werden Informationen über die E-Mail übermittelt. Der Header enthält z.B. Informationen über den Weg, den die Nachricht durch das Internet genommen hat. Einige Informationen des Headers werden von dem E-Mail-Programm ausgelesen und Ihnen angezeigt. Dies sind u.a. der Absender und die Priorität. Die Header-Informationen setzen sich aus einem Schlüsselwort und dem Wert dieses Schlüssels zusammen. So ist im obigen Beispiel `from` das Schlüsselwort und die E-Mail-Adresse der Wert.

Sie ergänzen die Funktion durch einen Absender, indem Sie in die Klammer `"from: xyz@yxz.de"` schreiben.

Datum- und Zeitfunktionen

`date()`

Diese Funktion gibt eine Zeitangabe formatiert zurück. Je nach der Zeitangabe, die Sie wünschen, setzen Sie unterschiedliche Platzhalter ein, z.B. `d` für Tag (in der Schreibweise 01 etc.), `M` für eine dreistellige Monatsangabe, `Y` für eine vierstellige Jahreszahl: `date("d M Y");` gibt also das aktuelle Datum in folgender Schreibweise zurück: `tt mmm yyyy`.

Einen Überblick über die Platzhalter finden Sie in der Tabelle 3.4:

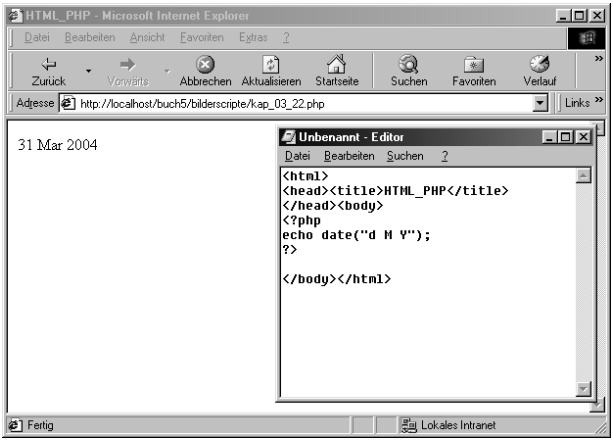

Bild 3.22: Die Verwendung der Datumsfunktion und die Ausgabe im Browser

Platzhalter	Ausgabe
G??	Stunden im Format: 1 – 12
G	Stunden im Format: 0 – 23
H??	Stunde im Format: 01 – 12
H	Stunde im Format: 00 – 23
I??	Minuten im Format: 00 – 59
D??	Tag des Monats 01-31
D	Tag der Woche in dreistelliger Abkürzung (Mon)
J??	Tag des Monats ohne führende Null
W??	Numerische Darstellung des Wochentags (0 = Sonntag)
M??	Monat im Format: 01 – 12
M	Monat als dreistellige Abkürzung
n??	Monat ohne führende Null
Y	Vierstellige Jahreszahl
y??	Zweistellige Jahreszahl
a??	am oder pm

Tabelle 3.4: Ein Überblick über die Platzhalter für die Datumsformate??Alle Angaben mit ?? sind klein zu schreiben??

`checkdate()`

Damit können Sie ein Datum auf Gültigkeit überprüfen lassen. Das Ergebnis der Prüfung kann `true` (wahr) oder `false` (falsch) sein. In die Klammer schreiben Sie numerisch den Monat, den Tag und das Jahr:

```
$checkdatum =checkdate(12, 24, 2001);
```

`time()`

Diese Funktion gibt die aktuelle Zeit des Servers zurück. Bei `time()` bleibt die Klammer leer. Wenn Sie `echo time()` schreiben, erhalten Sie den aktuellen Unix-Zeitstempel zurück. Dieser enthält die Anzahl der Sekunden seit Beginn der Unix-Epoche, also seit dem 01.01.1970, 00.00.00 Uhr. Ihr Browser gibt also eine ziemlich krause Zahl aus.

Mathematische Funktionen

`abs()`

Diese Funktion gibt von einer bestimmten Zahl den Absolutwert zurück. Ist die Zahl ohnehin ein Absolutwert, passiert nichts.

`max()`

Mit `max` wird der numerisch größte Eingabewert zurückgegeben. Es müssen mindestens zwei Parameter übergeben werden. Werfen Sie einen Blick auf das Bild 3.23.

`min()`

Mit `min` wird der numerisch kleinste Eingabewert zurückgegeben. Es müssen mindestens zwei Parameter übergeben werden.

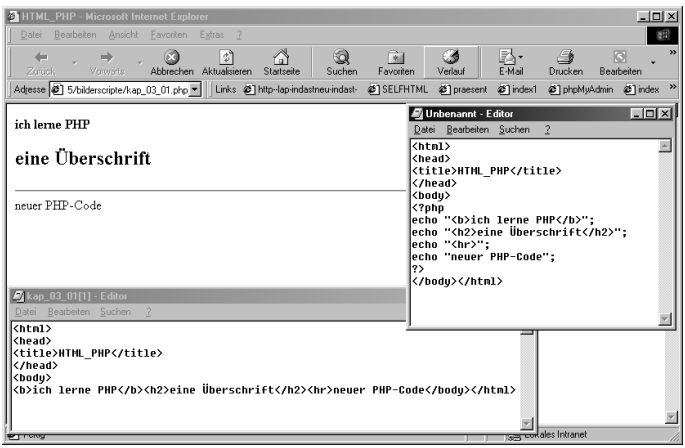

Bild 3.23: Die Funktion `max()` *gibt den größten Wert zurück.*

Dateisystem-Funktionen

`fopen()`

Diese Funktion öffnet eine Datei oder einen URL. Wenn die Datei noch nicht existiert, wird sie mit `fopen` erzeugt. In der Klammer steht der Name der Datei und der Modus (Dateizeiger).

`fgets()`

Mit dieser Funktion lesen Sie eine Zeile einer Datei aus, und zwar von der Position des Dateizeigers.

`fputs()`

Mit `fputs` schreibt man Daten an die Position des Dateizeigers. Sie können stattdessen auch `fwrite` benutzen.

`filesize()`

Diese Funktion liefert die Größe einer Datei.

MySQL-Funktionen

`mysql_connect()`

Diese Funktion setzen Sie ein, um eine Verbindung zum Datenbank-Server herzustellen. Als Argumente setzen Sie in die Klammer *hostname*, *Benutzername* und *Kennwort*.

`mysql_create_db()`

Diese Funktion erzeugt (bei entsprechender Berechtigung) eine neue Datenbank auf dem Server.

`mysql_close()`

Diese Funktion schließt eine Verbindung zum Datenbank-Server.

`mysql_db_query()`

Diese Funktion sendet ein SQL-Statement an die Datenbank, die dieses abarbeitet.

> **Hinweis**
>
> Die Funktionen des Dateisystems und die MySQL-Funktionen werden an dieser Stelle nur kurz erwähnt, aber nicht weiter erläutert, da sie im konkreten Kontext wesentlich verständlicher werden. ■■■hier verweis auf Kapitel mit MYSQL+Dateisystem Nutzung, glaube Gästebuch oder Counter■■■

Kontrollstrukturen

Ohne die Anwendung von Kontrollstrukturen kommen Sie beim Programmieren nicht aus. Hinter der Bezeichnung *Kontrollstrukturen* verbergen sich Bedingungen und Schleifen. Mit einer **Bedingung** ist gemeint, dass man PHP anweist, etwas zu vergleichen und dann in Abhängigkeit von dem Resultat des Vergleichs

bzw. der Prüfung die eine oder andere Aktion durchführt. Erst mit dem Einsatz dieser Möglichkeit kreieren Sie wirklich dynamische Seiten, denn es erfolgen unterschiedliche Reaktionen je nach vorausgegangenen »Ereignissen«.

PHP kennt zwei Bedingungen: die `if`-Anweisung und die `switch`-Anweisung.

Die if-Anweisung

Diese Anweisung benutzen Sie, wenn Sie im Klartext sagen möchten: Wenn eine Bedingung, die mithilfe eines Ausdrucks formuliert wird, erfüllt ist, also `true` (wahr) ergibt, dann soll der angegebene Befehlsblock ausgeführt werden. Ist der Ausdruck `false` (falsch), wird dieser Programmblock ignoriert und die Programmausführung mit dem nachfolgenden Befehl fortgesetzt. Eine if-Anweisung wird folgendermaßen geschrieben:

```
if(Bedingung) {was soll geschehen;}
```

In der runden Klammer steht der logische Ausdruck, in der geschweiften Klammer, auch als Block bezeichnet, der Dann-Teil der Anweisung bzw. Anweisungen. Jede Anweisung muss mit Semikolon abgeschlossen werden, aber Sie können im Prinzip beliebig viele Anweisungen in den Dann-Teil schreiben. Am Ende folgt die geschlossene geschweifte Klammer. Das folgende Listing zeigt einen Ausschnitt eines Scripts mit einer if-Anweisung.

```
<?php
if($kontakt=="e-mail")
{
echo "geben Sie eine e-mail-Adresse ein";
}
?>
```

Listing 3.6: Ein Scriptfragment mit einer if-Bedingung

Hier wurde (z.B. durch eine Option eines Auswahlfeldes in einem Formular) eine Kontaktoption (e-mail) übergeben, die in der Variablen `$kontakt` gespeichert ist.

Grundsätzlich kann die if-Bedingung weiter verschachtelt werden, d.h. jeder Block kann erneut einen if-Befehl enthalten. Dies führt aber – wie Sie sich denken können – zu ziemlich unübersichtlichen Codes und ist deshalb nicht unbedingt empfehlenswert. In der Regel gibt es Alternativen.

Vergleichsoperatoren und logische Operatoren

Für den richtigen Gebrauch der if-Anweisung müssen Sie noch einiges wissen. Wenn Sie als Bedingung einfach einen zuvor definierten Variablennamen verwenden, sagen Sie damit quasi so etwas wie: Wenn die Variable existiert und wenn sie einen Wert hat, der nicht Null ist, ergibt die Prüfung `true`. Anders bei einem Vergleich. Wenn Sie eine if-Anweisung dazu einsetzen, um zu prüfen, ob

sie mit einem spezifischen Wert übereinstimmt, dann brauchen Sie einen Vergleichsoperator. In diesem Fall ist dies nicht einfach das Gleichheitszeichen (mit dem Sie einer Variablen einen Wert zuweisen), sondern es sind zwei Gleichheitszeichen hintereinander ==. Es ist wichtig, sich daran zu erinnern:

```
if($anrede == "Frau")
```

müsste es also heißen, wenn eine Anweisung ausgeführt werden soll, sofern als Anrede *Frau* übergeben und der Wert in der Variablen `$anrede` gespeichert ist. Ist dies der Fall, ergibt die Prüfung der Bedingung `true` (wahr).

Achtung

Doppelachtung! Wenn Sie in einer if-Bedingung die doppelten Gleichheitszeichen vergessen, gibt PHP keine Fehlermeldung aus, sondern nimmt die mit dem einfachen Gleichheitszeichen angewiesene Zuweisung vor. Bei der Zuweisung ändert sich der Wert der Variablen.

Nach `if($anrede = "Frau")` steht in `$anrede` der Wert *Frau*.

Diese Fehler sind schwer zu finden und zeigen keine typischen Auswirkungen auf das Script, sodass man nicht sagen kann: Immer, wenn das und das passiert, haben Sie ein doppeltes Gleichheitszeichen in der if-Bedingung vergessen. Der einzige Hinweis, den wir Ihnen geben können, ist der Folgende:

Gibt das Script keinen Fehler aus und scheint der Code eigentlich richtig zu sein, aber die Ausgabe des Scripts ergibt partout nicht das, was Sie sich gedacht haben, dann kontrollieren Sie die doppelten Gleichheitszeichen.

Andere Vergleichsoperatoren sind Ihnen wahrscheinlich bekannt. So können Sie in einer if-Anweisung die mathematischen Zeichen für Größer als und Kleiner als benutzen, also > und < bzw. >= und <=, beispielsweise so:

```
if($kosten < 50) {echo "Zahlen Sie bitte per Scheck";}
```

Einen Überblick über die Operatoren bietet die Tabelle 3.5:

Symbol	Bedeutung
==	Gleichheit
!=	Ungleichheit
>	Größer als

Tabelle 3.5: Vergleichsoperatoren

Symbol	Bedeutung
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich

Tabelle 3.5: Vergleichsoperatoren (Forts.)

Vermutlich kennen Sie noch aus der Schule auch die speziellen (logischen) Operatoren, mit denen Ausdrücke verknüpft werden. Dies sind:

- `AND`
- `OR`
- `XOR`

Sie können diese Operatoren bei `if`-Bedingungen einsetzen. Wenn Sie beispielsweise `$variable1 AND $variable2` schreiben, bedeutet dies, dass nur `wahr` zurückgegeben wird, wenn beide Variablen wahr sind. Bei `OR` reicht es, wenn eine der Variablen wahr ist, um als Ergebnis `wahr` auszugeben. Mit `XOR` ergibt die Prüfung `falsch`, wenn beide Variablen gleich sind.

Verwendung von if-else

Neben der reinen if-Anweisung kommt sehr häufig die nächste logische Bedingung ins Spiel: die if-Anweisung, erweitert durch `else` (sonst). Mit der else-Klausel wird ein Code ausgeführt, wenn die Prüfung der Bedingung `false` ergeben hat. Die Syntax ist Folgende:

```
if (Bedingung) {Anweisung1;}
else {Anweisung2;}
```

Das würde dann in einem Script beispielsweise so aussehen:

```
if($kosten < 50) {echo "Zahlen Sie bitte per Scheck";}
else {echo "geben Sie Ihr Konto an";}
```

Neben der else-Klausel kennt PHP auch die `elseif`-Klausel. Damit können mehrere Ausdrücke (Bedingungen) geprüft werden, indem in dem else-Zweig noch ein weiterer if-Befehl eingebaut wird. Sie verwenden die if-Anweisung mit elseif-Klausel folgendermaßen:

```
if (Bedingung1) {Anweisung1;}
elseif (Bedingung2) {Anweisung2;}
else {Anweisung3;}
```

Den else-Teil können Sie auch weglassen, wenn es keine Anweisung für den Fall gibt, dass die Prüfung der Bedingungen nicht `true` zurückgegeben hat.

1. Öffnen Sie gegebenenfalls den Editor und beginnen Sie ein neues Script mit `<?php`.
2. In der nächsten Zeile verwenden Sie die Funktion `date()`, die je nach Formatierungsanweisung, die in der Klammer steht, ein Datum in formatierter Form zurückgibt. *W* ist das Symbol für die numerische Darstellung des Wochentags. Schreiben Sie in die nächsten Zeilen:

```
$tag=date("w");
if($tag==0 OR $tag==6)
```

3. Dann beginnt der Anweisungsblock. Sie schreiben:

```
{echo "endlich Wochenende";}
```

4. Nun beginnt die elseif-Klausel:

```
elseif ($tag==5) {echo "fast geschafft";}
```

5. Als Letztes schreiben Sie die else-Anweisung, danach schließen Sie den PHP-Teil.

```
else {echo "Arbeiten";}
?>
```

6. Im Ganzen sieht das Script folgendermaßen aus:

```
<?php
$tag=date("w");
if($tag==0 OR $tag==6) {echo "endlich Wochenende";}
elseif ($tag==5) {echo "fast geschafft";}
else {echo "Arbeiten!";}
?>
```

Listing 3.7: Das Script mit einer `elseif`*-Klausel: Je nach Wochentag erscheint ein anderer Text.*

Die switch-Anweisung

In vielen Fällen erweist es sich als sinnvoll, statt der if-Anweisung die Alternative `switch` zu benutzen. Entgegen der if-elseif-Auwertung prüft die switch-

Bedingung immer nur eine Bedingung bzw. einen Ausdruck (meistens den Wert einer Variablen) und führt je nach Resultat des Vergleichs die angegebenen Befehle aus. Dabei wird nicht entweder `true` oder `false` zurückgegeben, sondern mit mehreren Fallunterscheidungen gearbeitet:

ergibt der Vergleich den Fall 1, dann Anweisung 1;
ergibt der Vergleich den Fall 2, dann Anweisung 2;
ergibt der Vergleich den Fall 3, dann Anweisung 3;

Sie verwenden die switch-Anweisung im Script folgendermaßen:

```
switch ($Variable)
{
case "wert1":
Anweisung1;
break;
case "wert2":
Anweisung2;
break;
case "wert3":
Anweisung3;
break;
default:
Anweisung4;
break;
}
```

PHP beginnt die Auswertung am Anfang; sobald PHP den Fall (`case`) findet, der mit dem Wert korrespondiert, wird die Anweisung ausgeführt, und zwar tapfer so lange, bis das Ende der switch-Anweisung erreicht ist oder bis es auf `break` stößt. Deswegen also das `break`. Aus Gründen der Konsistenz schreiben wir es auch nach der `default`-Anweisung, wobei die `default`-Anweisung allerdings optional ist.

In ***Kapitel 5 Währung umrechnen***, in dem beispielhaft ein Euroumrechner programmiert wird, verwenden wir die switch-Bedingung, sodass Sie dort sehen können, wie dieser Befehl konkret eingesetzt wird.

Schleifen

Schleifen sind ein Herzstück der Programmsteuerung. Mit ihnen wird angegeben, wie oft ein Programmstück ausgeführt bzw. wiederholt werden soll. Die erste Schleife, die Sie in PHP verwenden – die `while`-Anweisung – bewirkt, dass eine Anweisung wiederholt wird, so lange die Bedingung erfüllt ist, der Ausdruck also `true` zurückgibt. Der Ausdruck wird vor jedem Schleifendurchlauf geprüft. Die Struktur der while-Schleife ist vergleichbar mit der if-Anweisung.

Die Verwendung der while-Schleife

Die while-Schleife wird so lange ausgeführt, wie eine bestimmte Bedingung zutrifft. In der Regel arbeitet man bei der while-Schleife mit einem internen Wert, der zunächst in eine Variable geschrieben wird. Das schafft die Bedingung für die Schleife. Generell wird die while-Schleife folgendermaßen geschrieben:

```
while (Bedingung) {Anweisung;}
```

Probieren Sie die while-Anweisung anhand eines einfachen Scripts, indem Sie ein neues Dokument beginnen und die üblichen HTML-Tags eingeben.

Fügen Sie in den Body-Container die PHP-Tags `<?php` und `?>` ein.

Schreiben Sie dann zwischen die PHP-Tags:

```
$i=1
while($i<10)
{
echo ($i * $i). "<br>";
$i++;
}
```

Damit setzen Sie die Variable `$i` und weisen ihr den Wert 1 zu. Dann sagen Sie in der while-Schleife: So lange `$i` kleiner als 10 ist, soll der Wert mit sich selbst multipliziert werden; mit `echo` werden die Ergebnisse der Anweisung ausgegeben. Entscheidend ist die Anweisung `$i++`. Damit wird der Wert jeweils um eins hochgezählt oder inkrementiert. Der Prozess setzt sich fort, bis `$i` größer als 10 ist. An diesem Punkt endet die Schleife.

Speichern Sie das Dokument als PHP-Datei und rufen Sie es im Browser auf.

Die Verwendung der for-Anweisung

Während die while-Anweisung durchlaufen wird, bis der Ausdruck `false` ergibt, wird die for-Anweisung eingesetzt, um eine Anweisung so oft auszuführen, wie man es im Code angibt. Sie mögen die for-Anweisung von der Syntax her etwas komplizierter finden, dafür ist sie aber in gewisser Weise in der Formulierung der Bedingung einfacher als die while-Anweisung und deswegen auch wieder leichter zu handhaben. Sie werden feststellen, dass in manchen Fällen die while-Anweisung besser passt und in manchen Fällen die for-Anweisung. Die generelle Syntax ist die Folgende:

```
for (Variablenzuweisung/Schleifenbedingung1; Ausdruck2;Ausdruck3)
{
Anweisung;
}
```

Sie sehen, die Ausdrücke in den Klammern werden durch Semikolon getrennt. Der erste Ausdruck initiiert in der Regel die Variable, der zweite Ausdruck prüft, ob die Anweisungen ausgeführt werden oder nicht, der dritte Ausdruck wird immer dann ausgeführt, wenn die Prüfung der Bedingung `true` ergeben hat. In geschweiften Klammern folgt dann wie üblich der Block mit der/den Anweisungen.

Wir haben oben bereits über Arrays gesprochen. Eine von vielen Möglichkeiten, eine for-Anweisung zu benutzen, ist beispielsweise die Anweisung, alle Elemente eines Arrays ausgeben zu lassen. Probieren Sie die for-Anweisung in diesem Zusammenhang:

1. Öffnen Sie in Ihrem Texteditor ein neues Dokument.
2. Geben Sie die Standard-HTML-Tags sowie die PHP-Tags ein.
3. Erstellen Sie ein beliebiges Array oder orientieren Sie sich am Listing 3.8. Hier heißt die Array-Liste `$fotos`.
4. Nun schreiben Sie die `for`-Schleife:

```
for ($n = 0; $n<count($fotos); $n++)
{
echo "$fotos[$n]";
}
?>
```

Insgesamt sieht das Script folgendermaßen aus:

```
<html><head>
<title>Formulart-test</title>
</head><body>
<?PHP
$fotos[]="Sommer";
$fotos[]="Herbst";
$fotos[]="Winter";
for ($n=0; $n<count($fotos); $n++)
{
echo "$fotos[$n]<br>";
}
?>
</body></html>
```

Listing 3.8: Das Script mit einer for-Schleife, um auf die Elemente eines Arrays zuzugreifen

Speichern Sie das Script und öffnen Sie es im Browser. Das Ergebnis müsste dem Bild 3.24 entsprechen.

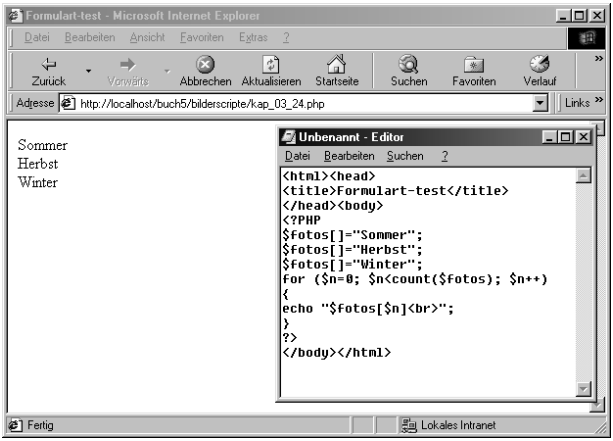

Bild 3.24: Die Ausgabe der Array-Elemente im Browser

Erläuterung der for-Schleife

Um das Verständnis für for-Schleifen etwas zu vertiefen, werden wir das Script aus Listing 3.8 noch einmal genauer betrachten. Zunächst wird das Array `$fotos` erstellt. Anschließend wird die Schleife durchlaufen.

Mit `for($n=0; $n<count($fotos); $n++)` sagen Sie PHP Folgendes: Setze die Variable `$n=0` für den ersten Durchlauf der Schleife. Prüfe bei jeden Schleifendurchlauf die Bedingung `$n<count($fotos)` – also ob `$n` kleiner ist als die Anzahl der Elemente im Array `$fotos`. Im Beispiel sind dies 3 Elemente. Mit `$n++` wird `$n` nach jedem Schleifendurchlauf um eins erhöht. Beachten Sie, dass diese Argumente der for-Schleife mit Semikolon getrennt werden. Wenn die formulierte Bedingung erfüllt ist (wahr), wird der Anweisungsblock durchlaufen: `{echo "$fotos[$n]<br>";}`.

Beim ersten Aufrufen der Schleife wird `$n=0` gesetzt, sodass die Bedingung erfüllt ist (0<3). Der Anweisungsblock wird ausgeführt – also `{echo "$fotos[0]<br>";}` – und somit das nullte Element des Arrays `$fotos` ausgegeben. Nachdem der Anweisungsblock abgearbeitet wurde, wird mit `$n++` `$n` um eins erhöht – `$n` ist dann also 1. Anschließend wird die Bedingung erneut geprüft. Sie ist wieder erfüllt, da 1<3, somit wird der Anweisungsblock erneut ausgeführt: `{echo "$fotos[1]<br>";}`. Wieder wird mit `$n++` `$n` um eins erhöht und anschließend die Bedingung getestet. Dieser Vorgang wiederholt sich so lange, bis die Bedingung nicht mehr erfüllt ist – also `$n` größer oder gleich `count($fotos)`ist. In diesem Fall werden die Befehle ausgeführt, die Sie im Script nach der Schleife angegeben haben.

Achten Sie darauf, die Schleifenbedingung so zu formulieren, dass sie auch irgendwann nicht mehr erfüllt wird. Eine for-Schleife in der folgenden Form würde unendlich oft durchlaufen werden:

```
for($n=0; $n>0; $n++).
```

Sie können den Schleifenzähler auch innerhalb des Anweisungsblocks der Schleife verändern. Das folgende Scriptfragment gibt nur jedes zweite Element eines Arrays aus, da der Zähler in der Schleife auch noch einmal erhöht wird.

```
for ($n=0; $n<count($fotos); $n++)
{
echo "$fotos[$n]<br>";
$n++;
}
```

Nicht nur den Schleifenzähler können Sie während des Anweisungsblocks verändern, sondern in Grenzen auch die Bedingung. Wenn Sie eine Schleife haben, die 10-mal durchlaufen werden soll, aber es einen Ausnahmefall gibt, bei dem die Schleife 30-Mal durchlaufen werden soll, können Sie diese Bedingung im Anweisungsblock einfügen. Die Schleife könnte dann so aussehen:

```
$max=10;
for ($n=0; $n<$max; $n++)
{
echo "$n <br>";
if($n==8){$max=30;}
}
```

Dieses kleine Beispiel ist nicht unbedingt sinnvoll, da `$n` immer den Wert 8 annimmt und damit die Bedingung der if-Anweisung erfüllt ist. Aber es geht hier darum, dass Sie sehen, dass auf die Bedingung auch während des Abarbeitens der Schleife Einfluss genommen werden kann.

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 Spitze des Eisberges abdecken 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 und zum Ersetzen von übereinstimmendem Text durch einen anderen Text.

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

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, brauchen 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 – z.B. entspricht der Punkt (.), der ein solches Zeichen ist, jedem einzelnen Buchstaben (». « gleich a, b, c, d, etc.) – bzw. nach bestimmten Wiederholungen eines Zeichens suchen.

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
.	jeder beliebiger Buchstabe
^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.

Tabelle 3.6: Metazeichen der regulären Ausdrücke

Zeichen	Übereinstimmung
a+	stimmt überein mit einem String, 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 (Forts.)

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 bbbb oder abc.

Die Kombination des Punktes . (steht dafür, dass ein beliebiger Buchstabe vorausgesetzt wird) mit dem Zeichen + ist ein Muster für einen beliebigen Buchstaben, und davon muss es mindestens einen geben. Insofern wird .+ als Zeichen häufig eingesetzt, denn es entspricht als Muster (mindestens) einem Buchstaben. 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 schiff gefolgt wird von (mindestens) einem weiteren f. *(schiff)+* hingegen entspricht nur einer Zeichenkette, die schiff und (mindestens) noch einmal schiff enthält. Alles klar?

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 die 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";
$string = "zu vergleichende Zeichenkette";
ereg($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 im letzten Kapitel des Buches, da wir Sie an dieser Stelle damit vermutlich überfordern würden. Im ***Kapitel 15 Tipps und Tricks*** finden Sie dann zwei Beispiele für die Verwendung regulärer Ausdrücke.