Teil I

Erste Schritte

Kapitel 1

PHP: Von der Homepage zum Portal

Willkommen bei PHP! Dieses Buch wird Sie mit nahezu allen Elementen der PHP-Sprache bekannt machen. Doch zuerst werden Sie das Produkt PHP kennen lernen – seine Geschichte, seine Merkmale und seine Zukunft.

In diesem Kapitel lernen Sie,

- ✘ was PHP ist,
- ✘ wie sich PHP entwickelt hat,
- ✘ welche Verbesserungen PHP 4 bringt,
- ✘ warum Sie PHP für Ihre Arbeit benutzen sollten.

1.1 Was ist PHP?

PHP (Personal Home Page) ist über ihren Namen hinausgewachsen. Ursprünglich sollte PHP als eine Gruppe von Makros die Pflege von privaten Homepages unterstützen. Seitdem wurde PHP immer wieder erweitert. So wurde aus einer Sammlung von Werkzeugen eine umfassende Programmiersprache, die große datenbankgestützte Websites verwalten kann.

Mit dem Funktionsumfang von PHP wuchs auch die Popularität dieser Sprache. Laut NetCraft (*http://www.netcraft.com*) lief PHP im November 1999 auf mehr als 1 Million Hosts. Im Oktober 2000 waren es bereits 3,8 Millionen Hosts. Laut E-Soft ist PHP das beliebteste Apache-Modul auf dem Markt, noch vor ModPerl.

Heute steht PHP offiziell für PHP HyperText Preprocessor. Es handelt sich um eine serverseitige Scriptsprache, die normalerweise in eine HTML-Datei eingebunden wird. Anders als eine gewöhnliche HTML-Seite wird ein PHP-Script nicht direkt vom Server an den Client geschickt. Es wird zuerst vom PHP-Modul geparst. Die HTML-Elemente im Script bleiben davon unberührt. Der PHP-Code wird interpretiert und ausgeführt. Mit PHP-Scripts können Datenbanken abgefragt, Bilder erzeugt, Dateien gelesen und geschrieben werden und es kann mit einem entfernten Server kommuniziert werden – es gibt unendlich viele Möglichkeiten. Die Kombination aus PHP-Programmausgabe und HTML wird an den Nutzer geschickt.

1.2 Die Entwicklung von PHP

Rasmus Lerdorf schuf 1994 mit einer Gruppe von Web-Publishing-Makros die erste Version von PHP. Diese wurden als Personal Homepage Tools veröffentlicht. Später wurden die Makros überarbeitet und um das Paket Form Interpreter (PHP/FI) erweitert. Aus Nutzersicht war bereits PHP/FI eine attraktive Lösung und wurde zunehmend beliebter. Auch die Entwicklergemeinde begann sich dafür zu interessieren. Seit 1997 arbeitet ein Programmierteam an diesem Projekt.

Die nächste Version, PHP 3 war ein Produkt dieser Gemeinschaftsarbeit. PHP 3 war eine erfolgreiche Neufassung von PHP mit einem neuen Parser sowie syntaktischen Änderungen und neuen Funktionen. Mit dieser Version wurde PHP zur begehrtesten Scriptsprache für die Server-Programmierung, und ihre Verbreitung war unaufhaltsam.

Ein Grund für die Beliebtheit von PHP ist die Unterstützung von Apache und MySQL. Weltweit ist Apache der am häufigsten eingesetzte Web-Server und PHP 3 kann als Apache-Modul kompiliert werden. MySQL ist eine leistungsstarke, frei erhältliche SQL-Datenbank. Für die Arbeit mit MySQL bietet PHP umfassende Funktionen. Die Kombination von Apache, MySQL und PHP ist unschlagbar.

Aber PHP unterstützt auch andere Umgebungen, Datenbanken, Server und Tools.

Mit der wachsenden Popularität von PHP änderte sich auch die Vorgehensweise im Bereich Web-Publishing. Mitte der 90er Jahre war es noch üblich, Sites mit statischen HTML-Seiten aufzubauen, selbst umfangreiche Websites mit Hunderten von Seiten. Jetzt werden immer häufiger die leistungsfähigen Funktionen von Datenbanken genutzt, um Inhalte effektiver zu verwalten und Websites zu personalisieren.

Es wird immer wichtiger, Datenbanken einzusetzen, um Inhalte zu speichern, und Scriptsprachen, um diese Daten aufzubereiten, weil die Daten von einer Quelle an immer mehr Stellen geschickt werden, z.B. an Mobiltelefone und PDAs, an das digitale Fernsehen und Internet-Umgebungen mit Breitbandübertragung.

In diesem Zusammenhang verwundert es nicht, dass PHP so beliebt ist.

1.3 Neu in PHP 4

PHP 4 bietet gegenüber PHP 3 zahlreiche neue Funktionen, die die Programmierung noch interessanter machen. Lassen Sie mich kurz einige davon vorstellen:

- Die neue Anweisung `foreach`, die ähnlich ist wie in Perl, vereinfacht den Array-Schleifendurchlauf. Wir werden diese Anweisung für die meisten Array-Beispiele in diesem Buch verwenden. Zusätzlich gibt es eine Vielzahl neuer Array-Funktionen, mit denen Arrays leichter bearbeitet werden können.
- Der Datentyp `Boolean`.
- Das PHP 3-Feature, Formular-Elemente wie Array-Elemente zu benennen, war bereits sehr hilfreich. Namen und Werte dieser Elemente sind dem Programm damit im Array-Format verfügbar. Dieses Feature wurde erweitert; jetzt werden auch mehrdimensionale Arrays unterstützt.
- PHP 4 unterstützt deutlich besser den objektorientierten Ansatz. Es ist z.B. jetzt möglich, eine überschriebene Methode in einer Unterklasse aufzurufen.
- PHP 4 bietet zur Realisierung von Benutzersitzungen systemeigene Unterstützung. Dafür können sowohl Cookies als auch die URL zur Datenübergabe eingesetzt werden. Sie können eine Variable für eine Sitzung »registrieren« und bei nachfolgenden Anfragen auf den Variablennamen und den Variablenwert zugreifen.
- Es wurde ein neuer Vergleichsoperator (===) eingeführt, der die Gleichheit von Typ und Wert prüft.
- Es gibt neue assoziative Arrays mit Server- und Umgebungsvariablen und eine Variable, die Information über die auf den Server übertragenen Dateien enthält.
- PHP 4 unterstützt Java und XML.

Diese und weitere Funktionen haben die Sprache enorm verbessert. Aber noch mehr ist unter der Oberfläche passiert.

1.4 Zend

Für PHP 3 wurde ein völlig neuer Parser entwickelt. Für PHP 4 wurde der Sprachkern (Script-Engine) geändert, wobei dies noch größere Auswirkungen hatte.

Zend ist der Sprachkern, auf dem PHP 4 nun basiert und der die Sprache deutlich leistungsfähiger gemacht hat.

Diese erhöhte Leistungsfähigkeit garantiert auch den künftigen Erfolg von PHP. Die meisten PHP 3-Scripts können auch ohne Änderung weiterhin ausgeführt werden, aber die neuen Scripts können bis zu zweihundertfach schneller ausgeführt werden.

Als kommerzielle Ergänzung zur Zend Engine sollen künftig PHP-Scripts kompiliert werden können, was die Performance noch einmal erhöhen wird.

Zend ist darauf ausgerichtet, die Performance zu steigern und mehr Flexibilität zu bieten. Die Integration von PHP als Modul in den Web-Server wurde verbessert, sodass das PHP-Modul für mehr Server-Typen geschrieben werden kann. Anders als ein CGI-Script, das nicht in den Server eingebettet wird, sondern mit jedem Zugriff als eigenständiges Programm gestartet wird, läuft ein Server-Modul zusammen mit dem Server. Dadurch wird die Performance verbessert, weil der PHP-Interpreter für die Ausführung einer PHP-Seite nicht gestartet werden muss.

1.5 Vorteile von PHP

Vieles spricht für PHP. Wenn Sie schon mit anderen Scriptsprachen gearbeitet haben, werden Sie feststellen, dass der Entwicklungsprozess mit PHP oft bedeutend schneller ist. Das Open Source-Produkt PHP wird von einem kompetenten Entwicklungsteam und einer engagierten Benutzergemeinschaft gut gepflegt. Darüber hinaus kann PHP auf den gängigen Betriebssystemen und den meisten Servern ausgeführt werden.

1.5.1 Schnelle Entwicklung

Sie werden feststellen, dass der Entwicklungsaufwand erheblich reduziert wird, weil Sie bei PHP HTML-Code und Script-Elemente trennen können. In vielen Fällen können Sie die Script-Erstellung vom Design und Aufbau der Seiten trennen. Das erleichtert die Programmierung und das Design kann effektiv und flexibel gestaltet werden.

1.5.2 Open Source

Für viele Leute bedeutet »Open Source« einfach, dass sie nichts bezahlen müssen, was natürlich schon ein Vorteil ist. So steht es auch auf der PHP-Site (`http://www.php.net`):

> *PHP kostet keinen Pfennig. Für alle, die nicht aus der UNIX-Welt kommen, mag das ungewöhnlich sein. Sie können PHP für kommerzielle und private Zwecke nutzen. Sie können es an Ihre Freunde weitergeben, Sie können es ausdrucken, an die Wand hängen oder zum Mittagessen verspeisen. Willkommen in der Welt der Open-Source-Software! Lächeln Sie, seien Sie glücklich, die Welt ist schön. Die vertraglichen Formalitäten können Sie in der offiziellen Lizenzvereinbarung nachlesen.*

Jedoch bieten gut gepflegte Open-Source-Projekte noch zusätzliche Vorteile. Sie profitieren von dem Erfahrungsschatz einer kommunikativen und engagierten Gemeinschaft. Mit großer Wahrscheinlichkeit können die bei Ihrer Programmierarbeit auftauchenden Schwierigkeiten nach einer kleinen Recherche schnell und einfach behoben werden. Falls Sie nichts finden, können Sie eine intelligente und fachkundige Lösung über eine Mailing-Liste erhalten.

Außerdem können Sie sicher sein, dass an Fehlern gearbeitet wird und bei Bedarf neue Features geliefert werden. Sie müssen nicht auf die neue kommerzielle Version warten, um von Verbesserungen profitieren zu können.

Es gibt kein eigennütziges Interesse an bestimmten Server-Produkten oder Betriebssystemen. Sie können wählen, was Sie oder Ihre Kunden brauchen. Sie können sicher sein, dass Ihr Programm läuft, egal für was Sie sich entscheiden.

1.5.3 Performance

Dank der leistungsfähigen Zend Engine schneidet PHP 4 im Vergleich mit ASP beim Benchmark-Test gut ab – manchmal ist es Testsieger. Kompiliertes PHP lässt ASP weit hinter sich.

1.5.4 Portierbarkeit

PHP kann auf vielen Betriebssystemen ausgeführt werden und unterstützt verschiedenste Server- und Datenbanktypen. Sie können für eine UNIX-Umgebung programmieren und Ihre Scripts problemlos auf NT übertragen. Sie können Ihr Projekt mit Personal Web Server testen und es unter UNIX installieren, wo PHP als Apache-Modul läuft.

1.6 Zusammenfassung

In diesem Kapitel stellten wir PHP vor. Sie haben erfahren, dass sich PHP von einer einfachen Makro-Gruppe zu einer funktionstüchtigen Script-Umgebung entwickelt hat. Sie haben etwas über PHP 4 und die Zend Script-Engine erfahren, über neue Features und verbesserte Leistung. Schließlich lernten Sie einige PHP-Features kennen, die PHP klar zur ersten Wahl unter den Web-Programmiersprachen machen.

1.7 Frage und Antwort

Frage: Ist PHP leicht zu lernen?

Antwort: Ja! Sie können die Grundlagen zu PHP in den 24 Kapiteln dieses Buchs lernen. PHP bietet eine Fülle von Funktionen, die Ihnen Dinge ermöglichen, die Sie in anderen Sprachen eigens programmieren müssten. PHP verwaltet für Sie Datentypen und Speicherangelegenheiten (vergleichbar mit Perl).

Die Kenntnis der Syntax und der Struktur einer Programmiersprache ist jedoch nur der Anfang. Letzten Endes erzielen Sie die besten Lernerfolge, indem Sie eigene Projekte entwickeln und aus Fehlern lernen. Betrachten Sie dieses Buch als Ausgangspunkt.

1.8 Workshop

Im Workshop werden Ihnen Fragen gestellt, die Ihnen helfen sollen, das Gelernte zu vertiefen. Versuchen Sie, die Testfragen zu beantworten, bevor Sie das nächste Kapitel lesen. Die Antworten zu den Fragen finden Sie in Anhang A.

1.8.1 Test

1. Was hat PHP ursprünglich bedeutet?
2. Wer hat die erste PHP-Version definiert?
3. Wie heißt die neue Script-Engine für PHP?
4. Nennen Sie ein neues PHP-Feature.

1.8.2 Übung

Sehen Sie sich den Aufbau des Buchs an. Überlegen Sie, wie Ihnen die behandelten Themen bei künftigen Projekten helfen könnten.

Kapitel 2

Installation

Bevor Sie mit PHP arbeiten können, müssen Sie einen PHP-Interpreter besorgen, installieren und konfigurieren. PHP ist für eine ganze Reihe von Plattformen und Server erhältlich.

In diesem Kapitel lernen Sie,

- ✘ welche Plattformen, Server und Datenbanken von PHP 4 unterstützt werden,
- ✘ wo Sie PHP und andere nützliche Open-Source-Software finden,
- ✘ wie Sie PHP auf Linux installieren,
- ✘ wie Sie Ihren PHP-Interpreter mit Features ausstatten,
- ✘ wie Konfigurationsanweisungen geschrieben werden,
- ✘ wie Sie Hilfe bekommen, wenn etwas schief geht.

2.1 Plattformen, Server, Datenbanken und PHP

PHP ist plattformübergreifend. Es läuft auf Windows-Betriebssystemen, den meisten UNIX-Versionen einschließlich Linux und sogar auf Macintosh. Es werden eine ganze Reihe von Web-Servern unterstützt. Dazu gehören Apache (ebenfalls Open Source und plattformübergreifend), Microsoft Internet Information Server, WebSite Pro, iPlanet Web-Server und Microsofts Personal Web Server. Mit Letzterem können Sie Ihre Scripts offline auf einem Windows-Rechner testen. Aber auch Apache läuft unter Windows.

Sie können PHP als Standalone-Anwendung kompilieren und es dann von der Befehlszeile aus aufrufen. In diesem Buch konzentrieren wir uns auf die Erstellung von Web-Anwendungen. Aber auch als allgemeine Script-Erstellungsumgebung darf PHP 4 nicht unterschätzt werden. Es ist durchaus mit Perl vergleichbar.

PHP lässt sich problemlos an Datenbanken anschließen. Diese Eigenschaft ist einer der Gründe, warum PHP für anspruchsvolle Web-Anwendungen eine ausgezeichnete Wahl ist. Manche Datenbanken werden direkt unterstützt, wie Adabas D, InterBase, Solid, dBASE, mSQL, Sybase, Empress, MySQL, Velocis, FilePro, Oracle, UNIX dmb, Informix und PostgreSQL. PHP unterstützt auch ODBC.

In diesem Buch werden wir mit einer Kombination aus Linux, Apache und MySQL arbeiten. Diese Komponenten können kostenlos aus dem Internet heruntergeladen und genutzt werden. Die Installation auf einem PC ist relativ einfach. Wo und wie Sie Linux für Ihren Rechner erhalten, erfahren Sie unter `http://www.linux.org/dist/index.html`. Informationen zur Installation von Linux auf Ihrem PowerPC (LinuxPPC) erhalten Sie unter `http://www.linuxppc.org`.

Die Datenbank MySQL, auf die sich die Buchbeispiele beziehen, gibt es unter `http://www.mysql.com`. Es gibt Versionen für verschiedene Betriebssysteme, unter anderem für UNIX, Windows und OS/2.

Sie können auch weiterhin mit Windows, NT oder MacOS arbeiten – schließlich ist PHP eine plattformübergreifende Scriptsprache.

2.2 Wichtige Adressen

PHP finden Sie unter `http://www.php.net/`. PHP 4 ist Open-Source-Software, das heißt, Sie brauchen für den Download keine Kreditkarte.

Die PHP Website ist eine ausgezeichnete Quelle für die PHP-Programmierung. Ein komplettes Handbuch kann online unter `http://www.php.net/manual/` gelesen werden. Es enthält nützliche Anmerkungen anderer PHP-ProgrammiererInnen. Das Handbuch liegt in verschiedenen Formaten zum Download bereit.

2.3 Installation unter Linux und Apache

In diesem Abschnitt werden wir erläutern, wie PHP 4 und Apache unter Linux installiert werden. Dieser Vorgang ist für alle UNIX-Betriebssysteme mehr oder weniger derselbe. Vielleicht finden Sie ein bereits kompiliertes und deshalb leicht zu installierendes PHP für Ihr System. Wenn Sie PHP selbst kompilieren, können Sie besser steuern, welche Features Ihr PHP-Interpreter enthält.

Vor der Installation sollten Sie sicherstellen, dass Sie als Root-Benutzer angemeldet sind. Wenn Sie keinen Zugriff auf die Root-Kennungen haben, bitten Sie Ihre Systemverwaltung, PHP zu installieren.

Es gibt zwei Möglichkeiten, ein Apache-PHP-Modul zu kompilieren. Sie können entweder Apache neu kompilieren und PHP statisch dazulinken oder Sie kompilieren PHP als Dynamic Shared Object (DSO). Wenn Ihre Apache-Version mit DSO-Unterstützung kompiliert wurde, können neue Module auch ohne Rekompilierung des Servers integriert werden. Das ist der einfachere Weg. Wie es geht, wird in diesem Abschnitt besprochen.

Um herauszufinden, ob Apache DSOs unterstützt, rufen Sie die Apache-Binärdatei (httpd) mit dem Argument -1 auf. Der Aufruf geht davon aus, dass der Server im Verzeichnis /www installiert wurde.

```
/www/bin/httpd -1
```

Es wird eine Liste von Modulen angezeigt. DSO wird unterstützt, wenn die Liste folgenden Eintrag enthält:

```
mod_so.c
```

Falls nicht, müssen Sie Apache neu kompilieren. Eine detaillierte Anleitung dazu erhalten Sie zusammen mit Apache.

Sie brauchen die aktuelle PHP 4-Version aus dem Internet. PHP 4 wird als tar-Datei archiviert, die mit gzip komprimiert wird. So entpacken Sie die Datei:

```
tar -xvzf php-4.0.3.tar.gz
```

Nach dem Entpacken wechseln Sie zum PHP 4-Distributionsverzeichnis:

```
cd ../php-4.0.3
```

Im Distributionsverzeichnis finden Sie das Script `configure`. Mit Argumenten können Sie steuern, welche Features Ihre PHP-Version enthält. Wir zei-

gen Ihnen einige nützliche Befehlszeilen-Argumente. Sie können auch eigene Argumente angeben. Am Ende des Kapitels behandeln wir die `configure`-Optionen.

```
./configure  enable-track-vars \
             with-gd \
             with-mysql \
             with-apxs=/www/bin/apxs
```

Die Pfadangabe für das Argument `--with-apxs` wird auf Ihrem System anders lauten. Es kann sein, dass `apxs` im gleichen Verzeichnis wie das Apache-Programm liegt.

Wenn das Script `configure` ausgeführt wurde, starten Sie das Programm `make`. Um den Befehl erfolgreich ausführen zu können, brauchen Sie einen C-Compiler.

```
make
make install
```

Diese Befehle schließen die Kompilierung und Installation von PHP 4 ab. Jetzt können Sie Apache konfigurieren und ausführen.

2.4 Optionen für Configure

Beim Aufruf von `Configure` haben wir einige Befehlszeilen-Argumente angegeben, die festlegen, welche Features Ihr PHP-Interpreter haben wird. Das `Configure`-Script liefert eine Liste der verfügbaren Optionen. Dazu geben Sie im PHP-Distributionsverzeichnis ein:

```
./configure --help
```

Die ausführliche Ausgabeliste können Sie in eine Datei schreiben, um sie in Ruhe zu lesen:

```
./configure --help > configoptions.txt
```

Wir betrachten hier einige Optionen, die im Buch benötigt werden.

2.4.1 --enable-track-vars

Diese Option füllt assoziative Arrays automatisch mit Werten, die durch `GET`- und `POST`-Anfragen oder in einem Cookie geliefert werden. Mehr zu Arrays finden Sie in Kapitel 7, »Arrays«, und zu HTTP-Anfragen in Kapitel 13, »Die Verbindung zur Außenwelt«. Diese Option sollten Sie bei der Konfiguration angeben.

2.4.2 --with-gd

`--with-gd` ermöglicht die Unterstützung der GD-Library. Wenn Sie diese Library installiert haben, können Sie dynamisch Gif- oder PNG-Bilder erzeugen. Mehr zu dynamisch erzeugten Bildern finden Sie in Kapitel 14, »Arbeiten mit Dynamischen Bildern«. Sie können auch den Pfad für Ihre GD-Library angeben:

```
--with-gd=/path/to/dir
```

2.4.3 --with-mysql

`--with-mysql` ermöglicht die Unterstützung von MySQL-Datenbanken. Wenn auf Ihrem System MySQL nicht im Standardverzeichnis installiert ist, müssen Sie den Pfad angeben:

```
--with-mysql=/path/to/dir
```

Wie Sie wissen, unterstützt PHP auch andere Datenbanken. Tabelle 2.1 zeigt für eine Auswahl von Datenbanken die zugehörige `configure`-Option.

Datenbank	configure-Option
Adabas D	`--with-adabas`
FilePro	`--with-filepro`
msql	`--with-msql`
informix	`--with-informix`
iODBC	`--with-iodbc`
OpenLink ODBC	`--with-openlink`
Oracle	`--with-oracle`
PostgreSQL	`--with-pgsql`
Solid	`--with-solid`
Sybase	`--with-sybase`
Sybase-CT	`--with-sybase-ct`
Velocis	`--with-velocis`
LDAP	`--with-ldap`

Tabelle 2.1: configure-Optionen für einige Datenbanken

2.5 Apache konfigurieren

Nachdem Sie PHP und Apache kompiliert haben, sollten Sie die Apache-Konfigurationsdatei `httpd.conf` überprüfen. Diese Datei finden Sie im Unterverzeichnis `conf` des Apache-Installationsverzeichnisses. Fügen Sie folgende Zeilen hinzu:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

Damit wird erreicht, dass der PHP-Interpreter Dateien mit der Dateierweiterung `.php` parst. Dateien mit der Erweiterung `.phps` werden als PHP-Quellen ausgegeben. Das heißt, der Quellcode wird in HTML konvertiert und farbig ausgezeichnet. Beim Debuggen von Scripts kann das sehr nützlich sein.

Sie können auch andere Erweiterungen vergeben. Sogar Dateien mit der Erweiterung `.html` gelten als PHP-Dateien, wenn Sie folgende Zeile in die Datei `httpd.conf` eintragen:

```
AddType application/x-httpd-php .html
```

Beachten Sie aber, dass dadurch Ihre Website langsamer wird, weil der PHP-Interpreter jede Seite mit dieser Erweiterung parst, bevor sie zum Nutzer gelangt.

Wenn PHP bereits installiert ist und Sie keinen Zugriff auf die Apache-Konfigurationsdateien haben, können Sie dennoch die Erweiterungen für PHP-Dateien festlegen. Dazu schreiben Sie eine `AddType`-Anweisung in die Datei `.htaccess`. Die Anweisung gilt im eigenen Verzeichnis und in allen Unterverzeichnissen. Sie funktioniert aber nur, wenn `AllowOverride` für das Verzeichnis entweder `FileInfo` oder `All` einstellt. Dies wiederum muss der Systemadministrator erledigen.

Standardmäßig ist der Dateiname für die Zugangskontrolldatei `.htaccess`. Er könnte jedoch geändert worden sein. Überprüfen Sie den Namen anhand der Anweisung `AcessFileName` in der Datei `httpd.conf`. Auch ohne Root-Kennung müssten Sie die Apache-Konfigurationsdateien lesen können.

Wenn Sie keine Root-Berechtigung haben, bietet Ihnen die Datei `.htaccess` eine ausgezeichnete Möglichkeit, Ihren Server anzupassen. Eine weitere Möglichkeit, auch für Nicht-Root-Benutzer den PHP-Interpreter zu konfigurieren, bietet die `php.ini`-Datei.

2.5.1 php.ini

Nachdem Sie PHP kompiliert oder installiert haben, können Sie nachträglich in der Datei `php.ini` das Verhalten des PHP-Interpreters ändern. Auf UNIX-Systemen liegt diese Datei standardmäßig unter `/usr/local/lib`, auf Windows-Systemen im Windows-Verzeichnis. Eine `php.ini`-Datei im aktuellen Arbeitsverzeichnis hat Vorrang vor einer Datei im Standardverzeichnis. Sie können also das Verhalten des PHP-Interpreters in jedem Verzeichnis anders gestalten.

In Ihrem Distributionsverzeichnis müsste eine `php.ini`-Beispieldatei mit Standardeinstellungen zu finden sein. Wenn keine `php.ini`-Datei benutzt wird, gelten diese Standardeinstellungen.

Für die meisten Beispiele in diesem Buch genügen die Standardeinstellungen. Kapitel 22, »Fehlersuche«, informiert Sie über einige Änderungen, die Sie vornehmen könnten.

Anweisungen in der `php.ini`-Datei bestehen aus einem Anweisungsnamen und einem Wert, getrennt durch ein Gleichheitszeichen. Leerräume werden ignoriert.

Falls PHP bereits installiert wurde, möchten Sie vielleicht die Einstellungen in der `php.ini`-Datei überprüfen. Wenn Sie kein Zugriffsrecht auf diese Datei haben, können Sie eine eigene Datei in Ihrem Script-Verzeichnis erstellen, die Vorrang vor der Standarddatei hat. Mit der Umgebungsvariablen PHPRC können Sie eine `php.ini`-Datei festlegen.

Einstellungen in der `php.ini`-Datei können Sie jederzeit ändern. Wenn der PHP-Interpreter als Apache-Modul läuft, müssen Sie den Server neu starten, damit Änderungen wirksam werden.

2.5.2 short_open_tag

Die Anweisung `short_open_tag` legt fest, ob PHP-Code mit den Symbolen `<?` beginnen und mit `?>` enden kann. Sollte diese Option ausgeschaltet sein, sehen Sie eine der folgenden Zeilen:

```
short_open_tag = Off
short_open_tag = False
short_open_tag = No
```

Um die Funktion einzuschalten, geben Sie eine der folgenden Zeilen ein:

```
short_open_tag = On
short_open_tag = True
short_open_tag = Yes
```

Mehr zu den PHP-Start- und Ende-Tags finden Sie in Kapitel 3, »Das erste Script«.

2.6 Anweisungen für Fehlermeldungen

Zu Diagnosezwecken sollten Sie die Ausgabe von Fehlermeldungen im Browser zulassen. Standardmäßig ist sie eingeschaltet:

```
display_errors = On
```

Die Stufe der Fehlermeldungen kann eingestellt werden. In Kapitel 22 werden wir die verfügbaren Optionen für die Anweisung `error_reporting` ausführlicher besprechen. Sie sollten jedoch jetzt schon Folgendes festlegen:

```
error_reporting = E_ALL & - E_NOTICE
```

Diese Einstellung ist die Standardeinstellung. So werden alle Fehler gemeldet, aber keine Warnungen. Warnungen können mit einigen PHP-Techniken in Konflikt geraten.

2.6.1 Variablen

Bei PHP können Sie auf Variablen aus `GET`-Anfragen, `POST`-Anfragen oder Cookies zugreifen. Sie können dies in der `php.ini`-Datei einstellen.

Die Anweisung `track_vars` erstellt assoziative Arrays aus den Ergebnissen einer HTTP-Anfrage. Standardmäßig ist diese Funktion eingeschaltet:

```
track_vars = On
```

Die Anweisung `register_globals` legt fest, ob Werte aus einer HTTP-Anfrage als globale Variablen verfügbar gemacht werden sollen. Viele Scripts in diesem Buch benötigen folgende Einstellung:

```
register_globals = On
```

2.7 Hilfe!

Im Internet gibt es immer Hilfe, besonders zu Problemen mit Open-Source-Software. Sie sollten jedoch nicht gleich auf die Schaltfläche »Absenden« klicken. Egal, wie unlösbar Ihr Problem mit der Installation, Konfiguration oder Programmierung zu sein scheint – Sie sind damit nicht allein. Ihre Frage wurde wahrscheinlich schon von jemandem beantwortet.

Wenn Sie nicht mehr weiterwissen, sollten Sie zuerst auf der offiziellen PHP-Site nachsehen (`http://www.php.net/`), vor allem im Handbuch (`http://www.php.net/manual`).

Falls Sie dort nichts finden, benutzen Sie die Suchfunktion der PHP-Site. Der Rat, den Sie brauchen, könnte in einer Pressemitteilung oder in einer FAQ-Datei stehen. Eine andere hervorragende Quelle mit Suchfunktion ist die PHP-Wissensdatenbank (`http://www.faqts.com/knowledge-base/index.phtml`).

Noch immer kein Glück? Links zu Mailing-List-Archiven mit Suchfunktion finden Sie unter `http://www.php.net/mailsearch.php`. Diese Archive sind eine riesige Informationsquelle, zu der viele Größen der PHP-Gemeinde beigetragen haben.

Sollte Ihr Problem immer noch nicht gelöst sein, könnte es der PHP-Gemeinde nützen, wenn Sie Ihr Problem darstellen.

Sie können an der PHP Mailing-Liste teilnehmen (`http://www.php.net/support.php`). Sie erhalten zwar sehr viele Mails, aus einigen können Sie aber auch eine Menge lernen. Wenn Sie PHP-Scripts schreiben wollen, sollten Sie zumindest an einer Digest-Liste teilnehmen, die zu Ihrem Problem passt. Dorthin können Sie Ihre Frage schicken.

Ihre Anfrage sollte möglichst viele Informationen enthalten (ohne in einen Roman auszuarten). Die folgenden Angaben sind sachdienlich:

- ✘ Betriebssystem
- ✘ Installierte PHP-Version
- ✘ Gewählte `configure`-Optionen
- ✘ Ausgaben der `configure`- oder `make`-Befehle, die vor einer fehlgeschlagenen Installation angezeigt wurden
- ✘ Code, der Probleme verursacht

Warum wurde so ausführlich auf das Thema Mailing-Liste eingegangen? Zum einen wird Ihnen die Recherche sehr nützlich sein, weil damit Probleme in der Regel schnell und effizient gelöst werden können. Simple Fragen an technische Listen werden oft mit dem Hinweis auf Archive beantwortet, dem Startpunkt für Recherchen.

Zum anderen ist eine Mailing-Liste nicht mit einem technischen Support-Zentrum gleichzusetzen. Für Antworten auf Ihre Fragen wird niemand bezahlt. Trotzdem haben Sie Zugriff auf einen ungeheuren Schatz an Wissen, einschließlich dem von PHP-EntwicklerInnen. Berechtigte Fragen werden

zusammen mit den Antworten archiviert, um anderen PHP-ProgrammiererInnen zu helfen.

Sie sollten keine Angst haben, Ihre Fragen abzuschicken. PHP-EntwicklerInnen sind ein höflicher und hilfreicher Menschenschlag. Mit der Veröffentlichung Ihres Problems können Sie anderen vielleicht helfen.

2.8 Zusammenfassung

PHP 4 ist Open-Source-Software. Offen bedeutet auch, dass sie hinsichtlich Server, Betriebssystem und Datenbank nicht festgelegt sind.

In diesem Kapitel haben Sie erfahren, wo Sie PHP und andere Open-Source-Software finden, die Sie für Ihren Server und Ihre Website benötigen. Sie haben gelernt, wie PHP als Apache-Modul unter Linux kompiliert wird. PHP-Interpreter für andere Betriebssysteme enthalten entsprechende Schrittanleitungen. Sie haben einige der `configure`-Optionen kennen gelernt, mit denen die unterstützten Features eingestellt werden können. Sie haben etwas über die `php.ini`-Datei und darin enthaltene Anweisungen gelernt. Und schließlich haben Sie etwas über Hilfequellen erfahren. Jetzt müssen Sie nur noch die Sprache selbst in den Griff bekommen.

2.9 Frage und Antwort

Frage: Die Installation von Linux und Apache wurde besprochen. Bedeutet das, dass ich dieses Buch nicht gebrauchen kann, wenn ich mit einem anderen Server und einem anderen Betriebssystem arbeite?

Antwort: Nein. Eine der Stärken von PHP ist, dass es auf unterschiedlichen Plattformen läuft. Wenn es bei der Installation von PHP auf Ihrem Betriebssystem oder mit Ihrem Server Probleme gibt, dann lesen Sie die Dateien, die in Ihrem PHP enthalten sind. Dort finden Sie umfassende Schrittanleitungen zur Installation. Sollten Sie immer noch Probleme haben, dann lesen Sie den »Hilfe«-Abschnitt in diesem Kapitel. In den dort erwähnten Online-Quellen finden Sie sicherlich die Antwort auf Ihr Problem.

2.10 Workshop

Im Workshop finden Sie Testfragen, die Ihnen helfen sollen, das Verständnis des Gelernten zu vertiefen. Erst wenn Sie die Antworten verstanden haben, sollten Sie das nächste Kapitel lesen. Die Antworten finden Sie in Anhang A.

2.10.1 Test

1. Wo finden Sie das PHP-Handbuch?
2. Wie erhalten Sie auf einem UNIX-Betriebssystem Hilfe zu den Konfigurationsmöglichkeiten (den Optionen, die Sie in das `configure`-Script Ihres PHP-Interpreters eintragen)?
3. Wie lautet der Standardname für die Apache-Konfigurationsdatei?
4. Welche Zeile muss in die Apache-Konfigurationsdatei eingefügt werden, damit die Dateierweiterung `.php` erkannt wird?
5. Wie heißt die PHP-Konfigurationsdatei?

2.10.2 Übung

Installieren Sie PHP. Falls es schon installiert ist, überprüfen Sie die Konfiguration in der Datei `php.ini`.

Kapitel 3

Das erste Script

Nach der Installation und Konfiguration ist es an der Zeit, PHP auszuprobieren. In diesem Kapitel wird das erste Script erstellt und dessen Syntax analysiert. Nach dieser Lektion sollten Sie in der Lage sein, ein Dokument mit HTML und PHP zu erstellen.

In diesem Kapitel lernen Sie,

- wie Sie ein PHP-Script erstellen, auf den Server übertragen und ausführen,
- wie Sie in einem Dokument HTML und PHP kombinieren,
- wie Sie mit Kommentaren übersichtlichen Code schreiben.

3.1 Unser erstes Script

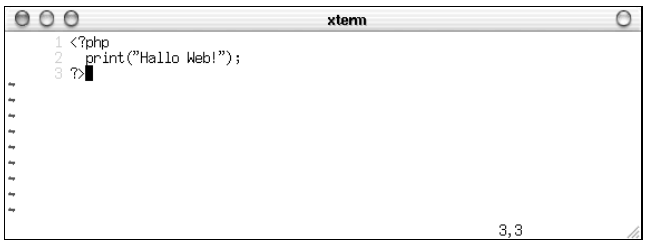
Wir steigen gleich mit einem PHP-Script ein. Als Erstes öffnen Sie einen Texteditor. PHP-Dateien sind, wie HTML-Dokumente, reiner Text. Sie können mit einem beliebigen Texteditor erstellt werden, wie Notepad unter Windows, Simple Text und BBEdit unter MacOS oder VI und Emacs unter UNIX-Betriebssystemen. Die gängigsten HTML-Editoren bieten zumindest ansatzweise PHP-Unterstützung.

Geben Sie den Quellcode aus Listing 3.1 ein und speichern Sie die Datei unter `first.php`.

Listing 3.1: Das erste PHP-Script

```
1: <?php
2:    print "Hallo Web!";
3: ?>
```

Abb. 3.1: Das erste Script als Text im Texteditor

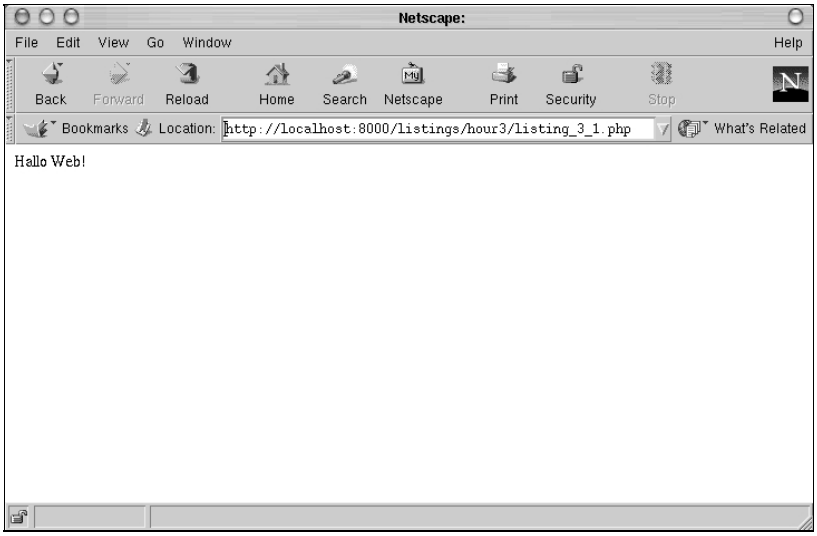


Durch die Dateierweiterung des PHP-Dokuments weiß der Server, dass er die Datei als PHP-Code behandeln muss, und ruft den Interpreter auf. Die Standarderweiterung für PHP 4-Dokumente ist `.php`. Dies kann jedoch in der Konfigurationsdatei des Servers geändert werden. Wie das geht, haben Sie in Kapitel 2, »Installation«, gesehen.

Wenn Sie nicht direkt am Server arbeiten, der Ihre PHP-Scripts zur Verfügung stellt, müssen Sie mit einem FTP-Client wie WS-FTP für Windows oder Fetch für MacOS Ihre Dokumente auf den Server übertragen.

Nach der Übertragung können Sie das Dokument im Browser aufrufen. Die Script-Ausgabe sollte zu sehen sein. Abbildung 3.2 zeigt die Ausgabe von `first.php`.

Abb. 3.2: Erfolgreiche Ausgabe von Listing 3.1 im Browserfenster



Sollte PHP nicht auf dem Server installiert sein oder die Dateierweiterung nicht erkannt werden, sieht die Ausgabe nicht wie in Abbildung 3.2 aus. Wahrscheinlich sehen Sie stattdessen den Quellcode. Abbildung 3.3 zeigt, was passiert, wenn die Dateierweiterung nicht erkannt wird.

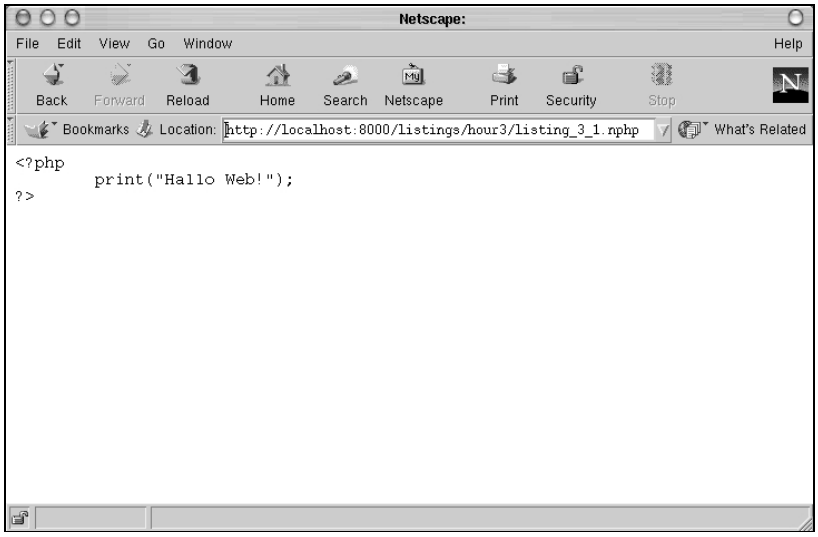


Abb. 3.3: Fehlerhafte Ausgabe aufgrund nicht erkannter Dateierweiterung

In diesem Fall überprüfen Sie zuerst die Dateierweiterung des PHP-Scripts. In Abbildung 3.3 wurde das Dokument versehentlich unter `listing_3_1.nphp` gespeichert. Stimmt die Dateierweiterung, sollten Sie überprüfen, ob PHP korrekt installiert wurde und ob der Server für die Dateierweiterung konfiguriert ist. In Kapitel 2 wurden die Installation und Konfiguration von PHP beschrieben.

Nachdem Sie Ihr Script übertragen und getestet haben, betrachten wir den Code etwas genauer.

3.1.1 Anfang und Ende von PHP-Code

Sie müssen dem Interpreter angeben, dass er Befehle ausführen soll. Wenn er es nicht weiß, wird der Code als HTML gelesen und als solcher im Browser ausgegeben. Tabelle 3.1 zeigt die vier Möglichkeiten, PHP-Code zu markieren.

Tabelle 3.2: Start- und Ende-Tags von PHP

Tag Style	Start-Tag	Ende-Tag
Standard-Tags	<?php	?>
Kurze Tags	<?	?>
ASP-Tags	<%	%>
Script-Tags	<SCRIPT LANGUAGE="php">	</SCRIPT>

Nur die Standard- und die Script-Tags werden zuverlässig in jeder Konfiguration erkannt. Die Erkennung kurzer Tags und ASP-Tags muss ausdrücklich in der `php.ini` eingeschaltet werden. In Kapitel 2 haben Sie die `php.ini` kennen gelernt.

Um die Erkennung kurzer Tags zu aktivieren, muss die Anweisung in der `php.ini` entsprechend lauten:

```
short_open_tag = On;
```

Standardmäßig sind kurze Tags eingeschaltet, sodass Sie `php.ini` nur editieren müssen, wenn Sie diese Option ausschalten wollen.

Um die Erkennung für ASP-Tags zu aktivieren, muss die entsprechende Anweisung eingeschaltet werden:

```
asp_tags = On;
```

Nachdem Sie `php.ini` editiert haben, können Sie eine der vier Markierungsarten für Ihre Scripts wählen – je nach Vorliebe. Wenn Sie jedoch mit XML arbeiten wollen, sollten Sie die kurzen Tags (`<? ?>`) ausschalten und mit den Standard-Tags (`<?php?>`) arbeiten.

Sehen wir uns die korrekten Schreibweisen für Listing 3.1 an. Sie können einen der vier Start- und Ende-Tags auswählen:

```
<?
print("Hallo Web!");
?>

<?php
print(Hallo Web!");
?>

<%
print("Hallo Web!");
%>

<SCRIPT LANGUAGE="php">
print("Hallo Web!");
</SCRIPT>
```

Einzeiliger Code kann in dieselbe Zeile wie Start- und Ende-Tag geschrieben werden:

```
<? print("Hallo Web!"); ?>
```

Jetzt wissen Sie, wie PHP-Code markiert wird. Sehen wir uns nun den Code selbst an (Listing 3.1).

3.1.2 Die Funktion print ()

`print()` gibt Daten aus, meistens im Browserfenster. Eine Funktion ist ein Befehl, der eine Aktion ausführt, die durch übergebene Daten gesteuert wird. Daten, die an diese Funktion übergeben werden, stehen meistens in Klammern nach dem Namen der Funktion. In unserem Fall wurde der Funktion eine Reihe von Zeichen, eine Zeichenkette, übergeben. Zeichenketten müssen zwischen einfachen oder doppelten Anführungszeichen stehen.

Funktionsaufrufe erfordern in der Regel Klammern nach dem Namen, egal, ob sie Datenübergabe fordern oder nicht. `print()` ist eine Ausnahme; hier ist das Setzen von Klammern optional. Üblicherweise werden sie weggelassen, weshalb wir es auch in unseren Beispielen so handhaben.

Die Code-Zeile in Listing 3.1 endet mit einem Strichpunkt. Damit erkennt der Interpreter, dass die Anweisung zu Ende ist.

Eine Anweisung richtet sich an den Interpreter. Im weitesten Sinne erfüllt sie für PHP den Zweck, den für uns ein gesprochener oder geschriebener Satz erfüllt. Eine Anweisung sollte mit einem Strichpunkt enden; ein Satz sollte mit einem Punkt enden. Zu den Ausnahmen gehören Anweisungen, die andere Anweisungen enthalten, und Anweisungen, die PHP-Code-Block beenden. In der Regel aber verursachen fehlende abschließende Strichpunkte einen Fehler.

Da die Anweisung in Listing 3.1 am Ende des PHP-Codes steht, ist der Strichpunkt optional.

3.2 Kombination von HTML und PHP

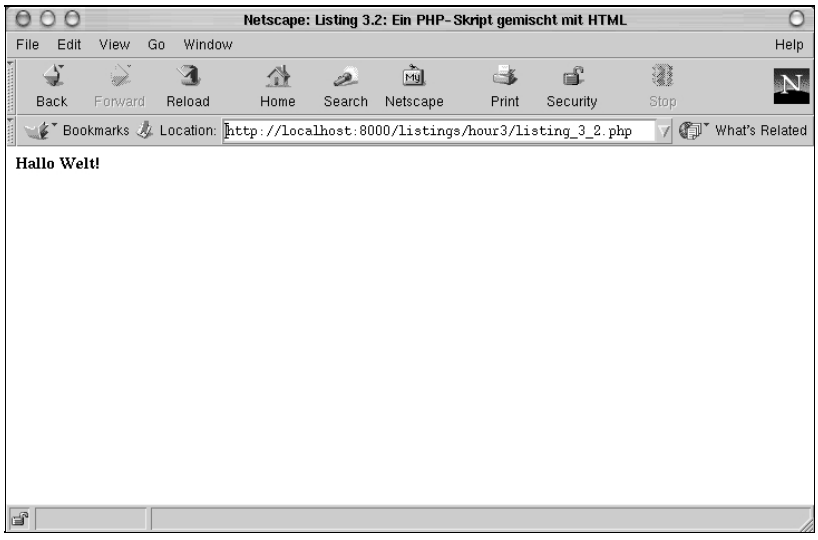
Das Script in Listing 3.1 ist reines PHP. Sie können es in ein HTML-Dokument integrieren, indem Sie HTML außerhalb des PHP-Start- und Ende-Tags angeben (siehe Beispiel 3.2).

Listing 3.2: PHP-Script in HTML

```
 1: <html>
 2: <head>
 3:   <title>Listing 3.2: Ein PHP-Skript gemischt mit HTML</title>
 4: </head>
 5: <body>
 6: <b>
 7: <?php
 8:   print("Hallo Welt!");
 9: ?>
10: </b>
11: </body>
12: </html>
```

Wie Sie sehen, ist die Verbindung von HTML und PHP leicht zu bewerkstelligen. Der PHP-Interpreter ignoriert alles, was sich außerhalb des PHP-Start- und Ende-Tags befindet. Die Ausgabe von Listing 3.2 im Browserfenster (Abb. 3.4) zeigt »Hallo Welt!« in Fettdruck. Schauen Sie sich den Quellcode des Dokuments an (Abb. 3.5) und Sie werden sehen, dass er sich nicht von einem normalen HTML-Dokument unterscheidet.

Abb. 3.4: Die Ausgabe von Listing 3.2 im Browserfenster



Sie können beliebig viel PHP-Code in ein Dokument einfügen und dazwischen nach Bedarf HTML schreiben. Obwohl PHP-Code an mehreren Stellen im Dokument steht, ist dies ein einziges Script. Alle Festlegungen im ersten PHP-Code (z.B. Variablen, Funktionen, Klassen) sind in der Regel für alle folgenden PHP-Abschnitte verfügbar.

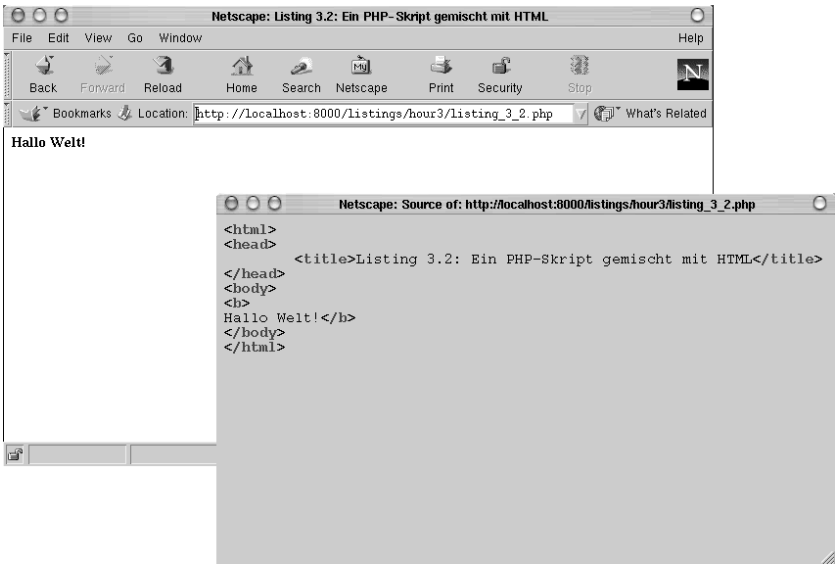


Abb. 3.5: Die Anzeige von Listing 3.2 als Quellcode im Browserfenster

3.3 Kommentare

Zum Zeitpunkt des Schreibens ist Ihnen der Code noch klar, aber vielleicht nicht mehr sechs Monate später, wenn Sie ihn überarbeiten wollen. Kommentare können Ihnen später viel Zeit ersparen und erleichtern anderen ProgrammiererInnen die Arbeit an Ihrem Code.

Ein Kommentar ist scriptbegleitender Text, der vom Interpreter ignoriert wird. Kommentare erleichtern die Lesbarkeit eines Codes und dienen als Anmerkungen zum Script.

Einzeilige Kommentare beginnen mit zwei Schrägstrichen (//) oder einem Gatterzeichen (#). Text nach diesen Zeichen bis zum Ende der Zeile oder bis zum PHP-Ende-Tag wird ignoriert.

```
// Dies ist ein Kommentar
# Auch dies ist ein Kommentar
```

Mehrzeilige Kommentare beginnen mit einem Schrägstrich und einem Stern (`/*`) und enden mit einem Stern, gefolgt von einem Schrägstrich (`*/`).

```
/*
Dies ist ein Kommentar
Dieser Text wird vom
Interpreter nicht analysiert
*/
```

3.4 Zusammenfassung

Jetzt können Sie ein einfaches PHP-Script auf einem korrekt konfigurierten Server ausführen.

Sie haben das erste PHP-Script geschrieben. Sie können mit einem Texteditor ein PHP-Dokument erstellen und richtig benennen. Sie haben vier Tag-Paare kennen gelernt, die Start und Ende von PHP-Code markieren. Sie haben gelernt, wie Sie mit der Funktion `print()` Daten im Browserfenster ausgeben und HTML und PHP zusammen in ein Script einbinden. Schließlich lernten Sie noch etwas über Kommentare und wie sie zu schreiben sind.

3.5 Fragen und Antworten

Frage: Welche Start- und Ende-Tags werden empfohlen?

Antwort: Das hängt von persönlichen Vorlieben ab. Die Standard-Tags (`<?php ?>`) sind am sichersten. Kurze Tags sind standardmäßig eingeschaltet und haben den Vorteil, dass sie schneller geschrieben werden können.

Frage: Welche Editoren soll ich für die Erstellung von PHP-Code nicht benutzen?

Antwort: Benutzen Sie keine Textverarbeitungsprogramme, die den Text für den Druck formatieren, wie zum Beispiel Word. Selbst wenn Sie Dokumente als reinen Text speichern, können sich versteckte Zeichen in Ihren Code einschmuggeln.

Frage: Wann soll ich Kommentare schreiben?

Antwort: Auch das ist eine Frage des Stils. Kurze Scripts sind auch nach einem längeren Zeitraum selbst erklärend. Längere oder komplexere Programme sollten kommentiert werden. Das spart auf die Dauer oft Zeit und Frustration.

3.6 Workshop

Im Workshop finden Sie Testfragen, die Ihnen helfen sollen, das Verständnis des Gelernten zu vertiefen. Erst wenn Sie die Antworten verstanden haben, sollten Sie das nächste Kapitel lesen. Die Antworten finden Sie in Anhang A.

3.6.1 Test

1. Können Benutzer den Quellcode Ihres PHP-Scripts lesen?
2. Wie sehen die Standard-PHP-Begrenzungs-Tags aus?
3. Wie sehen die Begrenzungs-Tags in ASP-Style aus?
4. Wie sehen die Begrenzungs-Tags für PHP-Script aus?
5. Welche Funktion schreiben Sie, damit Zeichenketten im Browserfenster ausgegeben werden?

3.6.2 Übung

Üben Sie mehrmals das Erstellen, Übertragen und Ausführen von PHP-Scripts.

Teil II

Die Sprache

Kapitel 4

Die Bausteine

Dieses Kapitel befasst sich mit einigen Grundbestandteilen der Sprache.

Hierzu gibt es eine Menge zu sagen und wenn Sie noch nicht programmiert haben, werden Sie in diesem Kapitel mit sehr vielen neuen Informationen konfrontiert. Keine Angst – später können Sie hier noch einmal die unklaren Punkte nachschlagen. Sie müssen sich nicht alles sofort merken. Wichtiger ist, dass Sie es verstehen.

Wenn Sie bereits programmieren können, sollten Sie dieses Kapitel zumindest überfliegen. Sie finden hier einige PHP-spezifische Funktionen.

In diesem Kapitel lernen Sie,

- ✘ was Variablen sind und wie sie eingesetzt werden,
- ✘ wie Sie Variablen definieren und darauf zugreifen,
- ✘ welche Datentypen es gibt,
- ✘ welche Operatoren am häufigsten benutzt werden,
- ✘ wie Operatoren in Ausdrücken eingesetzt werden,
- ✘ wie Sie Konstanten definieren und einsetzen.

4.1 Variablen

Eine Variable ist ein Speicherbereich, dem Sie einen bestimmten Wert zuweisen können. Eine Variable besteht aus einem frei wählbaren Namen mit vorgestelltem Dollarzeichen ($). Der Variablenname kann aus Buchstaben, Zahlen und dem Unterstrich (_) bestehen. Er darf keine Leerzeichen oder nicht alphanumerischen Zeichen enthalten. Beispiele für gültige Variablen:

```
$a;
$ein_sehr_langer_variablen_name;
$2453;
$schlaefrigZZZZ
```

Denken Sie an den Strichpunkt, der das Ende der PHP-Anweisung markiert. Die Strichpunkte in dem Code-Fragment oben gehören nicht zu den Variablennamen.

Eine *Variable* hat einen bestimmten Datentyp. Dies können Zahlen, Zeichenketten, Objekte, Arrays oder ein Boolescher Wert sein. Der Inhalt einer Variable kann jederzeit geändert werden.

Wie Sie sehen, gibt es für die Namensvergabe unzählige Möglichkeiten, obwohl Sie wohl kaum Variablennamen sehen werden, die ausschließlich aus Zahlen bestehen. Sie deklarieren die Variable in Ihrem Script. Normalerweise gehören die Variablendeklaration und die Wertzuweisung in dieselbe Anweisung.

```
$num1 = 8;
$num2 = 23;
```

Hier werden zwei Variablen deklariert. Der Wert wird mit dem Zuweisungsoperator (=) zugewiesen. Mehr über Zuweisungen erfahren Sie später im Abschnitt über Operatoren und Ausdrücke. Nachdem den Variablen Werte zugewiesen wurden, können sie wie Werte behandelt werden. Das heißt,

```
print $num1;
```

ist äquivalent zu

```
print 8;
```

wenn die Variable `$num1` den Wert 8 hat.

4.1.1 Dynamische Variablen

Es gibt auch die Möglichkeit, einen Variablennamen in eine Variable zu speichern. Die Zuweisung

```
$benutzer = "bob";
```

ist äquivalent zu

```
$varname="benutzer";
$$varname = "bob";
```

Die Variable `$varname` enthält die Zeichenkette `"benutzer"`. `$$varname` ist ein Dollarzeichen, gefolgt vom Wert der Variable `$varname`. Dieser Wert ist die Zeichenkette `"benutzer"`. Zusammen ergibt dies `$benutzer`.

Sie können auch mit einer Zeichenkette-Konstante eine dynamische Variable definieren. Dazu müssen Sie den Variablennamen in geschweifte Klammern schreiben:

```
${"benutzer"} = "bob";
```

Auf den ersten Blick scheint dies nicht viel zu bringen. Zusammen mit einem Verkettungsoperator und einer Schleife (siehe Kapitel 5, »Flexibilität«) können damit jedoch beliebig viele dynamische Variablen benutzt werden.

Die Syntax für den Zugriff auf Variablen und dynamische Variablen ist dieselbe:

```
$benutzer ="bob";
print $benutzer;
```

ist äquivalent zu

```
$benutzer ="bob";
$varname="benutzer";
print $$varname;
```

Wenn Sie jedoch eine dynamische Variable innerhalb einer Zeichenkette (steht in Anführungszeichen) ausgeben wollen, müssen Sie den Interpreter unterstützen:

```
$benutzer="bob";
$varname="benutzer";
print "$$varname";
```

Die `print`-Anweisung gibt nicht `"bob"` im Browserfenster aus, wie zu erwarten wäre. Stattdessen werden die Zeichen `$` und `benutzer` zusammen

als `$benutzer` ausgegeben. Steht die Variable zwischen Anführungszeichen, ersetzt PHP die Variable durch ihren Wert. In diesem Fall ersetzt PHP `$varname` durch die Zeichenkette `benutzer`. Das erste Dollarzeichen bleibt stehen. Damit PHP erkennen kann, dass innerhalb einer Zeichenkette eine dynamische Variable vorkommt, muss die Variable in geschweifte Klammern geschrieben werden. Die `print`-Anweisung im folgenden Beispiel gibt `"bob"` aus, den Wert der Variable `$benutzer`:

```
$benutzer="bob";
$varname="benutzer";
print "${$varname}";
```

Listing 4.1 fasst die vorangehenden Beispiele zu einem Script zusammen. Die Variable `$benutzer` wird über den Wert einer dynamischen Variable initialisiert und verwendet.

Listing 4.1: Variablen dynamisch setzen und darauf zugreifen

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.1: Dynamisches Setzen und Zugreifen auf ↵
      Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $varname = "benutzer";
 8: $$varname = "bob";
 9:
10: // ist equivalent zu:
11: // $benutzer = "bob";
12: // ${"benutzer"} = "bob";
13:
14: print "$benutzer<br>";       // gibt "bob" aus
15: print $$varname;             // gibt "bob" aus
16: print "<br>";
17: print "${$varname}<br>";     // gibt "bob" aus
18: print "${"benutzer"}<br>";   // gibt "bob" aus
19: ?>
20: </body>
21: </html>
```

4.1.2 Verweise auf Variablen

Standardmäßig werden Variablen per Wertkopie zugewiesen. Das heißt, wenn Sie die Variable `$eineVariable` der Variable `$andereVariable` zuweisen, wird eine Kopie des in `$eineVariable` enthaltenen Wertes in `$andereVariable` gespeichert. Nachfolgende Änderungen des Wertes von `$eineVariable` haben keinen Einfluss auf den Inhalt von `$andere-Variable`, wie in Listing 4.2 gezeigt.

Listing 4.2: Zuweisung per Wertkopie

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.2: Zuweisen eines Werts zu einer ↵
      Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $eineVariable = 42;
 8: // eine Kopie des Inhalts von $eineVariable wird ↵
       $andereVariable
 9: // zugewiesen
10: $andereVariable = $eineVariable;
11: $eineVariable = 325;
12: print $andereVariable;       // gibt 42 aus
13: ?>
14: </body>
15: </html>
```

In diesem Beispiel wird die Variable `$eineVariable` mit dem Wert 42 initialisiert. Dann wird die Variable `$eineVariable` der Variable `$andere-Variable` zugewiesen. Eine Kopie des Wertes von `$eineVariable` wird in `$andereVariable` geschrieben. Die Änderung des Wertes von `$eine-Variable` zu 325 hat keinen Einfluss auf den Inhalt von `$andere-Variable`. Das sehen Sie daran, dass die `print`-Anweisung 42 im Browserfenster ausgibt.

In PHP 4 können Sie dieses Verhalten ändern, indem der Variablen `$andereVariable` eine Referenz auf `$eineVariable` zugewiesen wird, wie in Listing 4.3.

Listing 4.3: Zuweisung per Referenz

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.3: Zuweisen einer Referenz zu einer ↵
      Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $eineVariable = 42;
 8: // eine Referenz auf den Inhalt von $eineVariable wird ↵
       $andereVariable
 9: // zugewiesen
10: $andereVariable = &$eineVariable;
11: $eineVariable = 325;
12: print $andereVariable;       // gibt 325 aus
13: ?>
14: </body>
15: </html>
```

Listing 4.3 unterscheidet sich von Listing 4.2 nur durch ein Zeichen. Das `&`-Zeichen vor der Variable `$eineVariable` sorgt dafür, dass der Variablen

`$andereVariable` eine Referenz auf `$eineVariable` zugewiesen wird anstatt einer Kopie des Wertes. Jetzt wirken sich Änderungen in `$eineVariable` auch auf `$andereVariable` aus, das heißt, `$eineVariable` und `$andereVariable` haben den gleichen Wert.

Diese Technik vermeidet den Kopieraufwand und kann die Performance ein wenig verbessern. Allerdings fällt dieser Performance-Gewinn nur ins Gewicht, wenn Sie viele Variablenzuweisungen verwenden.

Zuweisung per Referenz ist neu in PHP 4.

4.2 Datentypen

Jeder Datentyp stellt andere Anforderungen an den Arbeitsspeicher. Auch müssen Datentypen zum Teil unterschiedlich behandelt werden, damit sie im Script manipuliert werden können. Deshalb muss in manchen Programmiersprachen schon im Voraus deklariert werden, welchen Datentyp eine Variable enthält. PHP 4 ist hier toleranter, das heißt, es bestimmt den Datentyp, wenn der Variablen Daten zugewiesen werden. Dies hat Vor- und Nachteile. Zum einen können die Variablen flexibel gehandhabt werden und einmal eine Zeichenkette und ein anderes Mal eine Zahl enthalten. Zum anderen kann dies in großen Scripts für Verwirrung sorgen, wenn Sie in einer Variable einen Datentyp erwarten, der Inhalt aber ein ganz anderer ist.

Tabelle 4.1 zeigt die vier in PHP 4 verfügbaren Datentypen.

Tabelle 4.1: Datentypen

Typ	Beispiel	Beschreibung
Integer	`5`	Ganzzahl
Double	`3.234`	Fließkommazahl
String	`"hallo"`	Zeichenkette
Boolean	`true`	Die speziellen Werte true oder false
Object		Siehe Kapitel 8, »Objekte«
Array		Siehe Kapitel 7, »Arrays«

Die Datentypen »Arrays« und »Objekte« werden in Kapitel 7 und 8 besprochen.

Mit der PHP 4-Funktion `gettype()` können Sie den Datentyp der Variablen überprüfen. Dazu schreiben Sie die Variablen in die Klammern des

Funktionsaufrufs. `gettype()` gibt den Typ als Zeichenkette aus. In Listing 4.4 werden einer einzigen Variable Werte von vier verschiedenen Datentypen zugewiesen und einzeln mit `gettype()` überprüft.

Funktionsaufrufe werden detaillierter in Kapitel 6, »Funktionen«, beschrieben.

Listing 4.4: Überprüfung des Variablentyps

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.4: Bestimmen des Typs einer ↵
      Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $test = 5;
 8: print gettype($test);    // integer
 9: print "<br>";
10: $test = "fünf";
11: print gettype($test);    // string
12: print "<br>";
13: $test = 5.0;
14: print gettype($test);    // double
15: print "<br>";
16: $test = true;
17: print gettype($test);    // boolean
18: print "<br>";
19: ?>
20: </body>
21: </html>
```

So sieht die Ausgabe des Scripts aus:

```
integer
string
double
boolean
```

Integer ist eine ganze Zahl, das heißt, eine Zahl ohne Dezimalstellen. *String* ist eine Reihe von Zeichen, eine Zeichenkette. Zeichenketten müssen in doppelten (") oder einfachen (') Anführungszeichen eingeschlossen werden. *Double* ist eine Fließkommazahl, das heißt, eine Zahl mit Dezimalstellen. *Boolean* kann nur zwei spezielle Werte haben, entweder `true` oder `false`.

Frühere PHP-Versionen kannten den Booleschen Datentyp nicht. Zwar wurde `true` eingesetzt, es wurde jedoch als Integerwert 1 gespeichert.

4.2.1 Datentypen mit settype() ändern

In PHP kann mit der Funktion `settype()` der Typ einer Variablen geändert werden. Die zu ändernde Variable und der Typ, in den sie geändert werden soll, werden in Klammern geschrieben und durch Komma getrennt. In Listing 4.5 wird eine Fließkommazahl in die in diesem Kapitel besprochenen vier Datentypen konvertiert.

Listing 4.5: Änderung des Variablentyps mit settype()

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.5: Setzen des Typs einer Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $test = 3.14;
 8: print gettype($test);    // double
 9: print " -- $test<br>";   // 3.14
10: settype($test, string);
11: print gettype($test);    // string
12: print " -- $test<br>";   // 3.14
13: settype($test, integer);
14: print gettype($test);    // integer
15: print " -- $test<br>";   // 3
16: settype($test, double);
17: print gettype($test);    // double
18: print " -- $test<br>";   // 3
19: settype($test, boolean);
20: print gettype($test);    // boolean
21: print " -- $test<br>";   // 1
22: ?>
23: </body>
24: </html>
```

Jedes Mal überprüfen wir mit `gettype()`, ob die Änderung vorgenommen wurde. Der Wert der Variable `$test` wird im Browserfenster ausgegeben. Bei der Konvertierung der Zeichen »3.14« in eine Ganzzahl gehen die Dezimalstellen verloren. Deshalb enthält `$test` auch nach der Rückkonvertierung in den Datentyp Double den Wert 3. Schließlich konvertieren wir `$test` in einen Booleschen Wert. Alle Zahlen außer 0 werden durch die Konvertierung `true`. Bei der Ausgabe in PHP wird `true` als 1 dargestellt, `false` als leeres Zeichen, das heißt, `$test` wird als 1 ausgegeben.

4.2.2 Typänderung durch Casting

Indem Sie den Namen eines Datentyps in Klammern vor die Variable schreiben, erstellen Sie eine bereits in den angegebenen Datentyp konver-

tierte Kopie des Variablenwertes. Der Hauptunterschied zwischen `settype()` und Casting besteht darin, dass Casting eine Kopie erstellt, ohne die Original-Variable zu ändern. Ein Beispiel dazu bietet Listing 4.6.

Listing 4.6: Casting einer Variable

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.6: Casting einer Variablen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $var = 3.14;
 8: $test = (double) $var;
 9: print gettype($test);    // double
10: print " -- $test<br>";   // 3.14
11: $test = (string) $var;
12: print gettype($test);    // string
13: print " -- $test<br>";   // 3.14
14: $test = (integer) $var;
15: print gettype($test);    // integer
16: print " -- $test<br>";   // 3
17: $test = (double) $var;
18: print gettype($test);    // double
19: print " -- $test<br>";   // 3.14
20: $test = (boolean) $var;
21: print gettype($test);    // boolean
22: print " -- $test<br>";   // 1
23: ?>
24: </body>
25: </html>
```

Die Variable `$test` behält durchgängig den Datentyp Double. Das Casting von `$test` erstellt eine Kopie des Wertes und konvertiert ihn in den angegebenen Typ. Der neue Wert wird anschließend in der Variable `$varname` gespeichert. Da wir mit einer Kopie von `$test` arbeiten, geht uns keine Information verloren (wie in Listing 4.5).

4.3 Operatoren und Ausdrücke

Sie wissen jetzt, wie Daten Variablen zugewiesen werden. Sie können den Datentyp einer Variable überprüfen und ändern. Eine Programmiersprache nützt jedoch wenig, wenn Sie die gespeicherten Daten nicht bearbeiten können. Mit Hilfe von Operatoren können aus einem oder mehreren Werten neue Werte erzeugt werden. Die mit Operatoren bearbeiteten Werte sind die Operanden.

Ein *Operator* ist ein Symbol oder eine Reihe von Symbolen, die in Verbindung mit Werten eine Aktion ausführen oder einen neuen Wert erzeugen.

Ein *Operand* ist ein Wert in Verbindung mit einem Operator. In der Regel werden zwei Operanden durch einen Operator verknüpft.

Zuerst wollen wir zwei Operanden mit einem Operator kombinieren, um einen neuen Wert zu erhalten:

```
4 + 5
```

4 und 5 sind Operanden. In Kombination mit dem Operator (+) wird der Wert 9 erzeugt. Der Operator steht meistens zwischen den beiden Operanden. Es gibt jedoch Ausnahmen, auf die wir später eingehen werden.

Die Kombination aus Operanden und einem Operator zur Erzeugung eines Ergebnisses heißt Ausdruck. Ein Ausdruck muss nicht notwendig einen Operator enthalten. In PHP gilt als Ausdruck alles, was einen Wert erzeugt. Dazu gehören Integer-Konstanten wie 654, Variablen wie `$benutzer` und Funktionen wie `gettype()`. Deshalb sind in dem Ausdruck (`4 + 5`) zwei Ausdrücke und ein Operator enthalten.

Ein *Ausdruck* ist eine Kombination aus Funktionen, Werten und Operatoren, die einen Wert erzeugt. Was wie ein Wert behandelt werden kann, ist ein Ausdruck.

4.3.1 Zuweisungsoperator

Den Zuweisungsoperator haben Sie bereits kennen gelernt. Er wird durch das Zeichen = symbolisiert. Der Zuweisungsoperator weist den Wert des rechten Operanden dem linken Operanden zu:

```
$name ="matt";
```

Die Variable `$name` enthält jetzt die Zeichenkette `"matt"`. Diese Konstruktion ist ein Ausdruck. Auf den ersten Blick sieht es so aus, als würde der Zuweisungsoperator nur den Namen der Variable `$name` ändern, ohne einen Wert zu erzeugen. Jedoch erzeugt ein Ausdruck mit dem Zuweisungsoperator immer eine Kopie des Wertes des rechten Operanden.

```
print ( $name = "matt" );
```

gibt also die Zeichenkette `"matt"` im Browserfenster aus und weist zusätzlich `"matt"` der Variable `$name` zu.

4.3.2 Arithmetische Operatoren

Arithmetische Operatoren tun genau das, was Sie von ihnen erwarten. Eine Liste dieser Operatoren sehen Sie in Tabelle 4.2. Der Additionsoperator addiert den rechten Operanden zum linken Operanden. Der Subtraktionsoperator subtrahiert den rechten Operanden vom linken. Der Divisionsoperator dividiert den linken Operanden durch den rechten. Der Multiplikationsoperator multipliziert den linken Operanden mit dem rechten. Der Modulo-Operator gibt den Rest einer Division zurück.

Tabelle 4.2: Arithmetische Operatoren

Operator	Beschreibung	Beispiel	Beispielergebnis
+	Addition	10+3	13
-	Subtraktion	10-3	7
/	Division	10/3	3.333333333333
*	Multiplikation	10*3	30
%	Modulo	10%3	1

4.3.3 Verkettungsoperator

Der Verkettungsoperator wird durch einen Punkt dargestellt. Er behandelt beide Operanden als Zeichenketten und hängt den rechten Operanden an den linken an. Der Ausdruck

```
"Hallo"." Welt"
```

erzeugt

```
"Hallo Welt"
```

Es spielt keine Rolle, zu welchem Datentyp die Operanden gehören. Sie werden wie Zeichenketten behandelt und immer als Zeichenketten ausgegeben.

4.3.4 Zusätzliche Zuweisungsoperatoren

Obwohl es nur einen Zuweisungsoperator gibt, bietet PHP 4 einige kombinierte Operatoren, die den linken Operanden umwandeln und ein Ergebnis ausgeben können. In der Regel ändern Operatoren nicht die Werte der Operanden. Zuweisungsoperatoren sind eine Ausnahme. Ein kombinierter Zuweisungsoperator besteht aus einem gängigen Operatorsymbol gefolgt vom Gleichheitszeichen. Kombinierte Zuweisungsoperatoren nehmen Ihnen Arbeit ab.

Ein Beispiel:

```
$x = 4;
$x += 4; // $x hat nun den Wert 8
```

ist äquivalent zu

```
$x = 4;
$x = $x + 4; // $x hat nun den Wert 8
```

Für jeden arithmetischen Operator und jeden Verkettungsoperator gibt es einen Zuweisungsoperator. In Tabelle 4.3 sehen Sie eine Liste der gängigsten Kombinationen.

Tabelle 4.3: Einige kombinierte Zuweisungsoperatoren

Operator	Beispiel	Äquivalent zu
+=	$x += 5	$x = $x + 5
-=	$x -= 5	$x = $x – 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= " test"	$x = $x." test"

In jedem Beispiel aus Tabelle 4.3 wird der Wert von `$x` jeweils durch den rechten Operanden geändert.

4.3.5 Vergleichsoperatoren

Vergleichsoperatoren vergleichen die Operanden. Bei einem erfolgreichen Vergleich geben sie den Booleschen Wert `true` aus, andernfalls `false`. Dieser Ausdruckstyp ist in Kontrollstrukturen wie `if`- und `while`-Anweisungen sehr nützlich. Diese besprechen wir in Kapitel 5.

Um zu testen, ob der in `$x` enthaltene Wert kleiner ist als 5, benutzen Sie den Kleiner-Als-Operator:

```
$x < 5
```

Wenn `$x` 3 enthält, ergibt dieser Ausdruck `true`. Wenn `$x` 7 enthält, resultiert der Ausdruck in `false`.

Tabelle 4.4 listet die Vergleichsoperatoren auf.

Operator	Beschreibung	True	Beispiel	Ergebnis
==	gleich	links ist gleich rechts	$x == 5	false
!=	ungleich	links ist ungleich rechts	$x != 5	true
===	identisch	links ist gleich rechts und beide gehören zum selben Typ	$x === 5	false
>	größer als	links ist größer als rechts	$x > 4	false
>=	größer als und gleich	links ist größer oder gleich rechts	$x >= 4	true
<	kleiner als	links ist kleiner als rechts	x < 4	false
<=	kleiner als oder gleich	links ist kleiner oder gleich rechts	$x <= 4	true

Tabelle 4.4: Vergleichsoperatoren

Diese Operatoren werden meist für Ganzzahlen oder Double benutzt, obwohl der Gleichheitsoperator auch zum Vergleich von Zeichenketten eingesetzt wird.

4.3.6 Zusammengesetzte Vergleichsausdrücke

Logische Operatoren verknüpfen die Booleschen Werte. Zum Beispiel ergibt der `or`-Operator `true`, wenn entweder der linke oder der rechte Operand `true` ist.

```
true || false
```

ergibt `true`.

Der `and`-Operator ergibt `true` nur, wenn sowohl der linke als auch der rechte Operand `true` ist.

```
true && false
```

ergibt `false`. Sie verwenden logische Operatoren, um zwei oder mehr Ausdrücke, die einen Booleschen Wert liefern, zu kombinieren.

```
( $x > 2 ) && ( $x < 15 )
```

ergibt `true`, wenn `$x` einen Wert enthält, der größer ist als 2 und kleiner als 15. Die Klammern wurden wegen der besseren Lesbarkeit hinzugefügt. In Tabelle 4.5 sehen Sie eine Liste der logischen Operatoren.

Tabelle 4.5: Logische Operatoren

Operator	Beschreibung	True	Beispiel	Ergebnis
`||`	Or	links oder rechts ist wahr	true || false	True
or	Or	links oder rechts ist wahr	true || false	True
xor	XOr	links oder rechts ist wahr, aber nicht beide	true || false	False
&&	And	links und rechts sind wahr	true && false	False
and	And	links und rechts sind wahr	true && false	False
!	Not	der einzelne Operand ist nicht wahr	! true	False

Wegen der Operator-Präzedenz gibt es zwei Schreibweisen für den `or`- und den `and`-Operator. Wir werden dies in einem späteren Abschnitt besprechen.

4.3.7 Automatische Inkrementierung und Dekrementierung

Integer-Variablen werden Sie oft inkrementieren oder dekrementieren müssen, zum Beispiel zum Zählen von Schleifenwiederholungen. Dafür kennen Sie bereits zwei Möglichkeiten. Sie könnten die in `$x` enthaltene Ganzzahl mit dem Additionsoperator hochzählen:

```
$x = $x + 1; // $x ist inkrementiert
```

oder mit einem kombinierten Zuweisungsoperator:

```
$x += 1; // $x ist inkrementiert
```

In beiden Fällen wird die resultierende Ganzzahl der Variablen `$x` zugewiesen. Ausdrücke dieser Art sind sehr häufig. Deshalb bietet PHP einige spezielle Operatoren, mit denen Sie die Integer-Konstante 1 zu einer Integer-Variable addieren oder davon subtrahieren können. Das Ergebnis wird der Variablen zugewiesen. Diese Operatoren sind als Post-Inkrement- und Post-Dekrementoperatoren bekannt. Der Post-Inkrementoperator besteht aus zwei Pluszeichen, die an den Variablennamen angehängt werden:

```
$x++; // $x ist inkrementiert
```

inkrementiert die Variable `$x` um 1. Zwei Minuszeichen dekrementieren die Variable:

```
$x--; // $x ist dekrementiert
```

Wenn Sie den Post-Inkrement- oder den Post-Dekrementoperator in Verbindung mit einem Vergleichsoperator verwenden, wird der Operand erst nach dem Vergleich geändert:

```
$x = 3;
$x++ < 4; // true
```

In diesem Beispiel enthält `$x` die Zahl 3. Der Kleiner-als-Operator vergleicht diese Zahl mit 4 und es wird `true` ausgegeben. Nach diesem Vergleich wird `$x` inkrementiert.

Unter gewissen Umständen möchten Sie die Variable in einem Ausdruck inkrementieren oder dekrementieren, bevor der Vergleich durchgeführt wird. Für diesen Zweck bietet PHP den Prä-Inkrement- und den Prä-Dekrement-Operator. Wenn Sie allein stehen, wirken beide Operatoren wie der Post-Inkrement- und der Post-Dekrementoperator. Sie werden durch Plus- bzw. Minussymbole vor der Variablen dargestellt:

```
++$x; // $x ist inkrementiert
--$x; // $x ist dekrementiert
```

Wenn diese Operatoren in einem Vergleichsausdruck vorkommen, wird die Variable vor dem Vergleich inkrementiert.

```
$x = 3;
++$x < 4; // false
```

Hier wird `$x` vor dem Vergleich mit 4 inkrementiert. Der Vergleichsausdruck ergibt `false`, weil 4 nicht kleiner als 4 ist.

4.3.8 Operator-Präzedenz

In Ausdrücken mit Operatoren wertet der Interpreter von links nach rechts aus. In zusammengesetzten Ausdrücken mit mehr als einem Operator ist die Sache etwas komplizierter. Ein einfaches Beispiel:

```
4 + 5
```

Dies ist eine einfache Formel. PHP addiert 4 und 5. Wie sieht es mit dem nächsten Beispiel aus?

```
4 + 5 * 2
```

Hier taucht schon das erste Problem auf. Wird hier 4 und 5 zu einer Summe addiert und mit 2 multipliziert und 18 ausgegeben? Oder wird 5 mit 2 multipliziert und 4 hinzuaddiert mit dem Ergebnis 14? Wenn von links nach rechts gelesen wird, gilt die erste Lösung. In PHP haben Operatoren verschiedene Präzedenz. Der Multiplikationsoperator hat Vorrang vor dem Additionsoperator. Deshalb ist die zweite Lösung richtig.

PHP kann durch Klammern gezwungen werden, die Addition vor der Multiplikation durchzuführen:

```
( 4 + 5 ) * 2
```

Abgesehen von der Präzedenz in zusammengesetzten Ausdrücken sollten Sie Klammern benutzen, damit Ihr Code übersichtlich wird und schwer auffindbare Fehler vermieden werden. In Tabelle 4.6 sehen Sie eine Liste der in diesem Kapitel besprochenen Operatoren, geordnet nach Vorrang (von oben nach unten).

Tabelle 4.6: Ausgewählte Operatoren nach Präzedenz geordnet

Operatoren
++ -- (*cast*)
/ * %
+ -
< <= => >
== === !=
&&
||
= += -= /= *= %= .=
and
xor
or

Wie Sie sehen, steht `or` in der Präzedenz unter `||` und `and` unter `&&`. Sie können die logischen Operatoren mit geringerer Präzedenz einsetzen, um die Reihenfolge der Abarbeitung innerhalb eines komplexen Ausdrucks zu ändern. Das ist nicht immer zu empfehlen. Die folgenden beiden Ausdrücke sind äquivalent, aber der zweite Ausdruck ist einfacher zu lesen:

```
$x and $y || $z
( $x && $y ) || $z
```

4.4 Konstanten

Variablen bieten eine flexible Art, Daten zu speichern. Ihr Wert und der zu speichernde Datentyp können jederzeit geändert werden. Wenn Sie jedoch mit einem Wert arbeiten wollen, der für das ganze Script gleich bleiben soll, können Sie eine Konstante definieren. Dazu benutzen Sie die PHP-Funktion `define()`. Nach der Definition kann die Konstante nicht mehr geändert

werden. Bei der Funktion `define()` werden der Name der Konstante und ihr Wert in Klammern nach dem Funktionsaufruf geschrieben:

```
define( "KONSTANTEN_NAME", 42 );
```

Der Wert kann nur eine Zahl oder eine Zeichenkette sein. Üblicherweise wird der Name der Konstante in Großbuchstaben geschrieben. Auf Konstanten wird nur über ihre Namen zugegriffen; hier ist kein Dollarzeichen nötig. In Listing 4.7. wird eine Konstante definiert und verwendet.

Listing 4.7: Definition einer Konstante

```
 1: <html>
 2: <head>
 3:   <title>Listing 4.7: Definieren einer Konstanten</title>
 4: </head>
 5: <body>
 6: <?php
 7: define("BENUTZER", "Gerald");
 8: print "Herzlich Willkommen ".BENUTZER;
 9: ?>
10: </body>
11: </html>
```

Bitte beachten Sie, dass wir mit dem Verkettungsoperator den in der Konstante festgelegten Wert mit der Zeichenkette `"Herzlich Willkommen"` verknüpft haben. Der Grund dafür ist, dass der Interpreter innerhalb von Anführungszeichen nicht zwischen Konstante und einer Zeichenkette unterscheiden kann.

4.4.1 Vordefinierte Konstanten

In PHP sind einige Konstanten bereits eingebaut. Die Konstante `_FILE_` ist der Name der Datei, die gerade gelesen wird. `_LINE_` ist die Zeilenanzahl der Datei. Diese Konstanten sind für Fehlermeldungen nützlich. Mit `PHP_Version` können Sie herausfinden, welche PHP-Version im Script verwendet wird. Nützlich ist diese Funktion, wenn Sie wollen, dass ein Script nur unter einer bestimmten PHP-Version läuft.

4.5 Zusammenfassung

In diesem Kapitel wurden einige grundlegende Merkmale von PHP besprochen. Sie wissen jetzt, was Variablen sind, und wie ihnen mit dem Zuweisungsoperator Werte zugewiesen werden. Sie haben etwas über dynamische Variablen erfahren. Auch die Zuweisung per Referenz statt durch Wertkopie wurde besprochen. Sie haben Operatoren kennen gelernt und gesehen, wie die wichtigsten davon zu Ausdrücken kombiniert werden. Schließlich lernten Sie noch, wie Konstanten definiert und verwendet werden.

4.6 Fragen und Antworten

Frage: Warum kann es nützlich sein, den Datentyp zu kennen, der in einer Variable enthalten ist?

Antwort: Der Datentyp einer Variable bestimmt oft, was machbar ist. So könnte es sein, dass Sie sicherstellen wollen, dass eine Variable den Datentyp Integer oder Double enthält, bevor Sie sie für mathematische Berechnungen einsetzen.

In Kapitel 16, »Arbeiten mit Daten« werden wir detaillierter auf solche Fälle eingehen.

Fragen: Sollte ich mich bei der Namensvergabe für Variablen an Konventionen halten?

Antwort: Ihr Ziel sollte es immer sein, Ihren Code leicht lesbar und verständlich zu schreiben. Ein Variablenname wie `$ab12345` ist wenig aussagekräftig und ist anfällig für Tippfehler. Vergeben Sie kurze und beschreibende Variablennamen.

Eine Variable mit dem Namen `$f` sagt Ihnen wahrscheinlich wenig, wenn Sie nach zweimonatiger Pause an Ihrem Code weiterarbeiten. Dagegen ist eine Variable mit dem Namen `$filename` durchaus aussagekräftig.

Frage: Soll ich die Wertigkeiten der Operatoren auswendig lernen?

Antwort: Es spricht nichts dagegen. Jedoch würde ich mir die Energien für nützlichere Aufgaben aufsparen. Wenn Sie in Ihren Ausdrücken Klammern verwenden, wird Ihr Code einfach zu lesen sein und Sie definieren gleichzeitig Ihre eigene Reihenfolge der Abarbeitung.

4.7 Workshop

Im Workshop finden Sie Testfragen, die Ihnen helfen sollen, das Verständnis des Gelernten zu vertiefen. Erst wenn Sie die Antworten verstanden haben, sollten Sie das nächste Kapitel lesen. Die Antworten finden Sie in Anhang A.

4.7.1 Test

1. Welche der folgenden Variablennamen sind ungültig?

```
$ein_wert_eingegeben_vom_benutzer
$666666xyz
$xyz666666
$_____zaehler___
$der erste
$datei-name
```

2. Wie können Sie die String-Variable, die in dem Zuweisungsausdruck

```
$meine_variable = "dynamisch";
```

   definiert wird, einsetzen, um eine »Variablen«-Variable zu bekommen, die den Wert 4 enthält. Wie können Sie auf diese neue Variable zugreifen?

3. Welche Ausgabe erhalten Sie durch die folgende Anweisung?

```
print gettype("4");
```

4. Welche Ausgaben erhalten Sie durch den folgenden Code?

```
$test_val = 5.4566;
settype( $test_val, "integer" );
print $test_val;
```

5. Welche der folgenden Anweisungen enthalten keine Ausdrücke?

```
4;
gettype(44);
5/12;
```

6. Welche der Anweisungen in Frage 5 enthält einen Operator?

7. Welchen Wert gibt der folgende Ausdruck aus?

```
5 < 2
```

   Zu welchem Datentyp gehört der ausgegebene Wert?

4.7.2 Übungen

1. Schreiben Sie ein Script, das mindestens 5 verschiedene Variablen enthält. Füllen Sie die Variablen mit verschiedenen Datentypen und benutzen Sie die Funktion `gettype()`, damit jeder Typ im Browserfenster ausgegeben wird.
2. Weisen Sie zwei Variablen Werte zu. Benutzen Sie Vergleichsoperatoren um zu testen, ob der erste Wert
   1. der gleiche wie der zweite,
   2. kleiner als der zweite,
   3. größer als der zweite,
   4. kleiner als oder gleich dem zweiten ist.

   Lassen Sie die Ergebnisse im Browerfenster anzeigen.

   Ändern Sie die Werte Ihrer Test-Variablen und rufen Sie das Script noch einmal auf.

Kapitel 5

Flexibilität

Die Scripts des letzten Kapitels waren sehr einfach. Dieselben Anweisungen werden bei jeder Skriptausführung in derselben Reihenfolge ausgeführt. Das lässt wenig Raum für Flexibilität.

Sie erfahren in diesem Kapitel, wie Sie Ihre Scripts an bestimmte Bedingungen anpassen können.

In diesem Kapitel lernen Sie,

- wie Sie mit der `if`-Anweisung den Code nur dann ausführen, wenn ein Vergleich `true` ergibt,
- wie Sie einen alternativen Code ausführen, wenn der Vergleich einer `if`-Anweisung `false` ergibt,
- wie Sie die `switch`-Anweisung einsetzen, um einen Code abhängig vom Rückgabewert eines Vergleichs auszuführen,
- wie Sie den Code mit der `while`-Anweisung wiederholt ausführen,
- wie Sie mit der `for`-Anweisung übersichtlichere Schleifen programmieren,
- wie Schleifen verlassen werden,
- wie Schleifen verschachtelt werden.

5.1 Verzweigungen

Die meisten Scripts werten Bedingungen aus und ändern ihr Verhalten entsprechend. Durch diese Möglichkeit werden Ihre PHP-Seiten dynamisch und die Ausgabe passt sich gegebenen Bedingungen an. Wie in den meisten Programmiersprachen wird dies in PHP 4 durch die `if`-Anweisung erreicht.

5.1.1 if-Anweisung

Die `if`-Anweisung wertet einen zwischen Klammern stehenden Ausdruck aus. Wenn dieser Ausdruck einen wahren Wert ergibt, wird ein Programmblock ausgeführt. Andernfalls wird dieser Block komplett übersprungen. Dadurch kann im Script auf beliebige Faktoren reagiert werden.

```
if ( Ausdruck )
   {
   // Anweisungen, die ausgeführt werden, wenn der Ausduck
   // wahr ist.
   }
```

In Listing 5.1 wird der Block mit der Print-Anweisung nur ausgeführt, wenn die Variable die Zeichenkette `"froh"` enthält.

Listing 5.1: if-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.1: Die If-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $stimmung = "froh";
 8: if($stimmung == "froh") {
 9:   print "Super, ich bin gut drauf!";
10: }
11: ?>
12: </body>
13: </html>
```

Der Vergleichsoperator == vergleicht die Variable `$stimmung` mit der Zeichenkette `"froh"`. Stimmen beide überein, ist der Ausdruck wahr und der nach der `if`-Anweisung stehende Code wird ausgeführt. In diesem Beispiel steht der Code in geschweiften Klammern. Eigentlich ist das nur bei mehreren Zeilen notwendig. Deshalb ist auch folgende Schreibweise korrekt:

```
if ( $stimmung == "froh" )
    print "Super, ich bin gut drauf!";
```

Wenn Sie den Wert der Variablen `$stimmung` in `"traurig"` ändern und das Script ausführen, ist der Ausdruck der `if`-Anweisung falsch und der nachfolgende Code wird ignoriert. Das Script zeigt keine Wirkung.

5.1.2 if-Anweisung mit else-Klausel

Bei Verwendung der `if`-Anweisung soll häufig ein alternativer Code ausgeführt werden, wenn der geprüfte Ausdruck `false` ergibt. Dazu fügen Sie an die `if`-Anweisung `else` an, gefolgt von weiterem PHP-Code:

```
if ( Ausdruck )
   {
   // Anweisungen, die ausgeführt werden, wenn der
   // Ausdruck wahr ist.
   }
else
   {
   // Anweisungen, die in allen anderen Fällen
   // ausgeführt werden.
   }
```

Listing 5.2 ergänzt Listing 5.1 um Default-Code, der ausgeführt wird, wenn `$stimmung` ungleich `"froh"` ist.

Listing 5.2: if-Anweisung mit else

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.2: Die If- und else-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $stimmung = "traurig";
 8: if($stimmung == "froh") {
 9:   print "Super, ich bin gut drauf!";
10: } else {
11:   print "Ich bin gar nicht froh, sondern $stimmung.";
12: }
13: ?>
14: </body>
15: </html>
```

`$stimmung` enthält die Zeichenkette `"traurig"`, die ungleich `"froh"` ist, sodass der Ausdruck in der `if`-Anweisung `false` ergibt. Deshalb wird der erste Block übersprungen. Der Block nach `else` wird ausgeführt und im Browserfenster wird `"Ich bin gar nicht froh, sondern traurig"` ausgegeben.

Bisher können Sie die Entweder-Oder-Entscheidung umsetzen. PHP 4 erlaubt auch kaskadierende Auswertung mehrerer Ausdrücke.

5.1.3 if-Anweisung mit elseif-Klausel

Mit `if-elseif-else` werden mehrere Ausdrücke ausgewertet, bevor der Default-Code ausgeführt wird:

```
if ( Ausdruck )
   {
   // Anweisungen, die ausgeführt werden, wenn der Ausduck
   // wahr ist.
   }
elseif ( ein anderer Ausdruck )
   {
   // Anweisungen, die ausgeführt werden, wenn der
   // vorherige Ausdruck nicht wahr ist, dieser aber
   // wahr ist.
else
   {
   // Anweisungen, die in allen anderen Fällen
   // ausgeführt werden.
   }
```

Ergibt der erste Ausdruck nicht `true`, wird der erste Block übersprungen. `Elseif` veranlasst die Auswertung eines weiteren Ausdrucks. Wenn dieser Ausdruck `true` ergibt, wird der zweite Block ausgeführt. Andernfalls wird der nach `else` stehende Block ausgeführt. Sie können beliebig viele `elseif`-Klauseln einfügen. Wenn Sie keine Default-Ausführung brauchen, können Sie den `else`-Abschnitt weglassen.

Listing 5.3 erweitert das vorige Beispiel um `elseif`.

Listing 5.3 Eine if-Anweisung mit else und elseif

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.3: Die If-, else- und elseif-↵
      Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $stimmung = "traurig";
 8: if($stimmung == "froh") {
 9:   print "Super, ich bin gut drauf!";
10: } elseif($stimmung == "traurig") {
11:   print "Ach, mach dir nichts draus.";
12: } else {
13:   print "Ich bin weder froh noch traurig sondern $stimmung.";
14: }
15: ?>
16: </body>
17: </html>
```

`$stimmung` enthält die Zeichenkette `"traurig"`, die ungleich `"froh"` ist. Also wird der erste Block übersprungen. Die `elseif`-Klausel vergleicht den Inhalt von `$stimmung` mit `"traurig"`. Der Vergleich ergibt `true`. Also wird der nachfolgende Block ausgeführt.

5.1.4 switch-Anweisung

Sie können den Programmfluss auch mit der `switch`-Anweisung, abhängig vom Ergebnis eines Ausdrucks, steuern. Zwischen `switch` und `if` gibt es einige grundlegende Unterschiede. `If` in Verbindung mit `elseif` vergleicht mehrere Ausdrücke, `switch` wertet nur einen Ausdruck aus und führt je nach Ergebnis einen zugehörigen Programmcode aus. Das Ergebnis muss einen einfachen Datentyp haben (Zahl, `string` oder Boolescher Wert). Der Ausdruck einer `if`-Anweisung kann entweder `true` oder `false` ergeben. Dagegen wird das Ergebnis eines Ausdrucks in einer `switch`-Anweisung mit mehreren Werten verglichen.

```
switch ( Ausdruck )
   {
   case ergebnis1:
     // Anweisungen, wenn Ausdruck ergebnis1 ergibt
     break;
   case ergebnis2:
     // Anweisungen, wenn Ausdruck ergebnis2 ergibt
     break;
   default:
     // Anweisungen, in allen anderen Fällen
   }
```

Der Ausdruck in einer `switch`-Anweisung ist oft eine Variable. Innerhalb einer `switch`-Anweisung kommen `case`-Anweisungen vor. Jede davon vergleicht einen Wert mit dem Ergebnis des Ausdrucks der `switch`-Anweisung. Sind Wert und Ergebnis des Ausdrucks gleich, wird der nach der `case`-Anweisung stehende Code ausgeführt. Die `break`-Anweisung beendet die Ausführung der `switch`-Anweisung. Ohne `break` wird der Ausdruck in der nächsten `case`-Anweisung geprüft. Die Überprüfung geht bis zur optionalen `default`-Anweisung.

Vergessen Sie nicht, `break` am Ende eines `case`-Blocks einzufügen. Ohne `break` werden die `case`-Anweisungen bis `default` durchlaufen. Das ist meistens unerwünscht.

Listing 5.4 erweitert mit `switch` die Funktionalität der `if`-Anweisung.

Listing 5.4: switch-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.4: Die switch-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $stimmung = "traurig";
 8: switch($stimmung) {
```

```
 9:   case "froh":
10:     print "Super, ich bin gut drauf!";
11:     break;
12:   case "traurig":
13:     print "Ach, mach dir nichts draus.";
14:     break;
15:   default:
16:     print "Ich bin weder froh noch traurig sondern  $stimmung.";
17: }
18: ?>
19: </body>
20: </html>
```

Die Variable `$stimmung` wird mit `"traurig"` initialisiert. Diese Variable ist der Ausdruck der `switch`-Anweisung. Die erste `case`-Anweisung vergleicht `"froh"` und den Wert von `$stimmung`. Zeichenkette und Wert sind ungleich, also wird die nächste `case`-Anweisung geprüft. Die Zeichenkette `"traurig"` und der Wert von `$stimmung` sind gleich, also wird der nachfolgende Code ausgeführt. Mit `break` endet die Programmausführung.

5.1.5 ?-Operator

Der dreiteilige ?- Operator ist der `if`-Anweisung ähnlich. Dem ?-Operator folgen zwei durch Strichpunkt getrennte Ausdrücke. Abhängig vom Vergleichsergebnis wird der Wert des zweiten oder des dritten Ausdrucks zurückgegeben:

```
( Ausdruck )
   ?Rückgabe, wenn Ausdruck wahr ist
   :Rückgabe, wenn Ausdruck falsch ist;
```

Ergibt der Vergleich `true`, wird das Ergebnis des zweiten Ausdrucks zurückgegeben, sonst das Ergebnis des dritten Ausdrucks. In Listing 5.5 wird der dreiteilige Operator benutzt, um den Wert einer Variablen abhängig von `$stimmung` zu setzen.

Listing 5.5: ?-Operator

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.5: Der ?-Operator</title>
 4: </head>
 5: <body>
 6: <?php
 7: $stimmung = "traurig";
 8: $text = ($stimmung == "froh") ? "Super, ich bin gut drauf!"
 9:         : "Ich bin gar nicht froh,  sondern $stimmung.";
10: print $text;
11: ?>
12: </body>
13: </html>
```

Der Wert von `$stimmung` ist `"traurig"`. `$stimmung` wird mit der Zeichenkette `"froh"` verglichen. Der Vergleich ergibt `false`, also wird das Ergebnis des dritten Ausdrucks zurückgegeben.

Der dreiteilige Operator kann unübersichtlich werden, ist jedoch nützlich, wenn Sie nur zwei Alternativen brauchen und einen kompakten Code schreiben wollen.

5.2 Schleifen

Bis jetzt haben Sie Fallunterscheidungen kennen gelernt. Sie können auch festlegen, wie oft ein Programmstück ausgeführt wird. Dafür benutzen Sie Schleifen. Eine Schleife wird so lange ausgeführt, bis eine Bedingung erfüllt ist oder ein Befehl die Schleife beendet.

5.2.1 while-Anweisung

Die Struktur einer `while`-Anweisung ähnelt der Struktur einer einfachen `if`-Anweisung:

```
while ( Ausdruck )
      {
      // irgendwelche Anweisungen
      }
```

Solange der Ausdruck einer `while`-Anweisung `true` ergibt, wird derselbe Block immer wieder ausgeführt. Innerhalb des Blocks muss eine Bedingung für den Ausdruck der `while`-Anweisung stehen, sonst wird die Schleife unendlich oft ausgeführt. In Listing 5.6 wird innerhalb von `while` der Wert eines Zählers mit 2 multipliziert und ausgegeben.

Listing 5.6: while-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.6: Die while-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $zaehler = 1;
 8: while($zaehler <= 12) {
 9:   print "$zaehler mal 2 ist ".($zaehler*2)."<br>";
10:   $zaehler++;
11: }
12: ?>
13: </body>
14: </html>
```

Die Variable `$zaehler` wird initialisiert. Die `while`-Anweisung wertet diese Variable aus. Solange die Zahl in `$zaehler` kleiner oder gleich 12 ist, wird die Schleife ausgeführt. Die Zahl wird mit zwei multipliziert und das Ergebnis wird im Browser ausgegeben. Dann wird der Wert von `$zaehler` hochgezählt. Das ist sehr wichtig. Wenn der Wert von `$zaehler` nicht geändert wird, ergibt der Ausdruck in `while` niemals `false` und die Schleife läuft endlos weiter.

5.2.2 do...while-Anweisung

Im Gegensatz zur `while`-Anweisung steht bei `do...while` die Bedingung für die Schleife am Ende des Blocks.

```
do   {
     // Anweisungen
     }
     while ( Ausdruck );
```

Am Ende einer `do...while`-Anweisung muss immer ein Strichpunkt stehen.

Die Anweisung ist nützlich, wenn Sie einen Block mindestens einmal ausführen wollen, selbst wenn die Auswertung `false` ergibt, wie in Listing 5.7.

Listing 5.7: do...while-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.7: Die do-while-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $num = 1;
 8: do {
 9:   print "Laufende Nummer: ".$num."<br>\n";
10:   $num++;
11: } while($num > 200 && $num < 400)
12: ?>
13: </body>
14: </html>
```

Die `do...while`-Anweisung prüft, ob die Variable `$num` einen Wert enthält, der größer als 200 und kleiner als 400 ist. `$num` wurde mit der Zahl 1 initialisiert, also ergibt der Ausdruck `false`. Da der Block vor der Auswertung ausgeführt wird, gibt die Anweisung im Browserfenster eine Zeile aus.

5.2.3 for-Anweisung

Mit einer `for`-Anweisung erreichen Sie dasselbe wie mit einer `while`-Anweisung. Andererseits ist die `for`-Anweisung übersichtlicher und sicherer und erzielt dieselbe Wirkung. In Listing 5.6 wurde eine Variable außerhalb der `while`-Anweisung initialisiert und anschließend mit der Variablen im Ausdruck der `while`-Anweisung verglichen. Dann wurde der Zähler inkrementiert. Mit `for` erreichen Sie das in einer einzigen Zeile. Code wird kompakter und Sie vergessen nicht so leicht, den Zähler zu inkrementieren, was eine Endlosschleife bewirken würde.

```
for ( Variablenzuweisung; Test-Ausdruck; Inkrementieren der
Variablen )
    {
    // Anweisungen
    }
```

Die in Klammern gesetzten Ausdrücke der `for`-Anweisung werden durch Strichpunkt getrennt. In der Regel initialisiert der erste Ausdruck die Zählervariable, der zweite Ausdruck wertet die Schleifenbedingung aus und der dritte Ausdruck inkrementiert den Zähler. In Listing 5.8 werden zwölf Zahlen mit 2 multipliziert (vgl. Listing 5.6).

Listing 5.8: for-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.8: Die for-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: for($zaehler=1; $zaehler<=12; $zaehler++) {
 8:   print "$zaehler mal 2 ist ".($zaehler*2)."<br>";
 9: }
10: ?>
11: </body>
12: </html
```

Das Ergebnis von Listing 5.6 und 5.8 ist dasselbe. In der `for`-Anweisung ist der Code jedoch kompakter. Hier ist die Schleifenbedingung sofort klar, weil `$zaehler` zu Beginn der Anweisung initialisiert und inkrementiert wird. Der erste Ausdruck initialisiert die Variable `$zaehler` mit 1. Der zweite Ausdruck prüft, ob `$zaehler` einen Wert enthält, der kleiner oder gleich 12 ist. Der letzte Ausdruck inkrementiert den Wert von `$zaehler`.

Wenn das Programm die `for`-Schleife erreicht, wird die Variable `$zaehler` initialisiert und die Bedingung wird ausgewertet. Wenn die Auswertung `true` ergibt, wird der nachfolgende Block ausgeführt. Der Wert von `$zaehler` wird hochgezählt und der Ausdruck erneut ausgewertet. Dieser Vorgang wird so lange durchgeführt, bis die Auswertung `false` ergibt.

5.2.4 Schleifenabbruch mit break

Die Anweisungen `while` und `for` enthalten einen Ausdruck zum Beenden einer Schleife. Mit `break` können darüber hinaus nach zusätzlichen Abfragen Schleifen beendet und Fehler vermieden werden. In Listing 5.9 wird die Zahl 4000 durch den Wert einer inkrementierten Variablen geteilt und das Ergebnis wird im Browserfenster ausgegeben.

Listing 5.9: for-Schleife mit Division von 4000 durch eine inkrementierte Variable

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.9: Die for-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: for($zaehler=1; $zaehler<=10; $zaehler++) {
 8:   $temp = 4000/$zaehler;
 9:   print "4000 geteilt durch $zaehler ist ... $temp<br>";
10: }
11: ?>
12: </body>
13: </html>
```

Hier wird die Variable `$zaehler` mit 1 initialisiert, dann wird geprüft, ob der Wert von `$zaehler` kleiner oder gleich 10 ist. Anschließend wird 4000 durch den Zählerwert dividiert und das Ergebnis wird im Browserfenster ausgegeben.

Wenn der Zählerwert auf einer Benutzereingabe basiert, kann der Wert eine Minuszahl oder sogar eine Zeichenkette sein. Nehmen wir den ersten Fall. Wenn `$zaehler` mit -4 initialisiert wird, dann wird bei der fünften Blockausführung 4000 durch 0 dividiert, was nicht definiert ist. Listing 5.10 zeigt, wie dies vermieden wird: Die Schleife wird beendet, sobald `$zaehler` den Wert 0 enthält.

Listing 5.10: break-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.10: Vorzeitiger Abbruch der ↵
      for-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $zaehler = -4;
 8: for( ; $zaehler<=10; $zaehler++) {
 9:   if($zaehler == 0)
10:     break;
11:   $temp = 4000/$zaehler;
12:   print "4000 geteilt durch $zaehler ist ... $temp<br>";
13: }
14: ?>
15: </body>
16: </html>
```

Wenn in PHP 4 eine Zahl durch 0 dividiert wird, resultiert daraus kein fataler Fehler. Stattdessen wird eine Meldung ausgegeben und die Programmausführung fortgesetzt.

Prüfen Sie den Wert der Variablen `$zaehler` mit einer `if`-Anweisung. Ergibt der Vergleich 0, beendet `break` die Block-Ausführung und der Programmfluss setzt nach der `while`-Anweisung wieder ein. Beachten Sie, dass die Variable `$zaehler` außerhalb der `for`-Anweisung initialisiert wurde. Dadurch wird eine Situation simuliert, in der der Wert von `$zaehler` aus einer Benutzereingabe oder einer Datenbank kommt.

Sie können Ausdrücke in einer `for`-Anweisung weglassen, aber nicht die Strichpunkte.

5.2.5 Überspringen von Wiederholungen mit continue

Die Anweisung `continue` beendet den aktuellen Durchlauf, aber nicht die komplette Schleife. Es wird sofort die nächste Wiederholung ausgeführt. Die Anweisung `break` wie in Listing 5.10 einzusetzen, ist übertrieben. Mit `continue` wird der Fehler, der durch die Division durch 0 entsteht, vermieden, ohne dass der Schleifendurchlauf abgebrochen wird (siehe Listing 5.11).

Listing 5.11: continue-Anweisung

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.11: Überspringen von Zählschritten der ⏎
      for-Anweisung</title>
 4: </head>
 5: <body>
 6: <?php
 7: $zaehler = -4;
 8: for( ; $zaehler<=10; $zaehler++) {
 9:   if($zaehler == 0)
10:     continue;
11:   $temp = 4000/$zaehler;
12:   print "4000 geteilt durch $zaehler ist ... $temp<br>";
13: }
14: ?>
15: </body>
16: </html>
```

Wir haben die Anweisung `break` durch `continue` ersetzt. Wenn die Variable `$zaehler` gleich 0 ist, wird die Wiederholung übersprungen und sofort die nächste ausgeführt.

Die Anweisungen `break` und `continue` können die Lesbarkeit des Codes erschweren. Schleifenanweisungen werden differenzierter, sind aber auch fehleranfälliger, deshalb sollten diese beiden Anweisungen sparsam eingesetzt werden.

5.2.6 Verschachtelte Schleifen

Schleifenanweisungen können andere Schleifenanweisungen enthalten. Dies ist für die Arbeit mit dynamischen HTML-Tabellen nützlich. In Listing 5.12 wird durch zwei `for`-Anweisungen eine Multiplikationstabelle im Browserfenster ausgegeben.

Listing 5.12: Verschachtelung von zwei for-Schleifen

```
 1: <html>
 2: <head>
 3:   <title>Listing 5.12: Verschachteln von for-Anweisungen</title>
 4: </head>
 5: <body>
 6: <?php
 7: print "<table border=\"1\">\n";
 8: for($y=1; $y<=12; $y++) {
 9:   print "<tr>\n";
10:   for($x=1; $x<=12; $x++) {
11:     print "\t<td>";
12:     print ($x*$y);
13:     print "</td>\n";
14:   }
15:   print "</tr>\n";
16: }
17: print "</table>";
18: ?>
19: </body>
20: </html>
```

Die äußere `for`-Anweisung initialisiert die Variable `$y` mit dem Wert 1. Es wird geprüft, ob `$y` kleiner oder gleich 12 ist und es wird der Inkrementierungswert festgesetzt. Bei jeder Wiederholung wird das HTML-Element `TR` (Tabellenreihe) dem Browser übergeben und die nächste `for`-Anweisung definiert. Diese innere Schleife initalisiert die Variable `$x` und definiert Ausdrücke ähnlich denen der äußeren Schleife. Bei jeder Wiederholung übergibt

die innere Schleife das HTML-Element `TD` (Tabellenzelle) und das Ergebnis der Multiplikation von `$x` mit `$y` an den Browser. Als Resultat sehen Sie eine einwandfrei formatierte Multiplikationstabelle.

5.3 Zusammenfassung

In diesem Kapitel lernten Sie Kontrollstrukturen kennen und wie Sie damit flexible und dynamische Scripts erstellen. Die meisten dieser Strukturen werden in diesem Buch immer wieder auftauchen.

Sie haben gelernt, eine `if`-Anweisung zu definieren und alternative Aktionen mit `elseif` und `else` auszuführen. Sie wissen, wie Sie mit der Anweisung `switch` mehrere Vergleiche durchführen und damit den Programmfluss ändern. Sie kennen die Schleifenanweisungen `while` und `for` und wissen, wie Sie `break` und `continue` einsetzen müssen, um Schleifendurchläufe abzubrechen oder Wiederholungen zu überspringen. Schließlich haben Sie gelernt, wie Schleifen verschachtelt werden und Sie haben ein typisches Anwendungsbeispiel gesehen.

5.4 Fragen und Antworten

Frage: Muss die Auswertung einer Kontrollstruktur einen Booleschen Wert ergeben?

Antwort: Letztendlich ja. Bei einem Vergleich wird jedoch Null, eine undefinierte Variable oder eine leere Zeichenkette für Vergleichszwecke in `false` umgewandelt. Alle anderen Werte ergeben `true`.

Frage: Muss der Code einer Kontrollanweisung immer in Klammern stehen?

Antwort: Wenn der Code nur aus einer Zeile besteht, sind Klammern nicht nötig.

Frage: Wurden in diesem Kapitel alle möglichen Schleifen besprochen?

Antwort: In Kapitel 7, »Arrays«, finden Sie noch die Anweisung `foreach`. Damit können Schleifen jedes Array-Element durchlaufen.

5.5 Workshop

Im Workshop finden Sie Testfragen, die Ihnen helfen sollen, das Verständnis des Gelernten zu vertiefen. Erst wenn Sie die Antworten verstanden haben, sollten Sie das nächste Kapitel lesen. Die Antworten finden Sie in Anhang A.

5.5.1 Test

1. Wie schreiben Sie eine `if`-Anweisung, damit die Zeichenkette `"Nachricht für junge Leute"` im Browserfenster ausgegeben wird, wenn die Integer-Variable `$alter` eine Zahl zwischen 18 und 35 ist? Wenn `$alter` einen anderen Wert enthält, soll `"Nachricht für alle"` im Browserfenster ausgegeben werden.
2. Wie würden Sie den Code der Frage 1 erweitern, damit die Zeichenkette `"Nachricht für Kinder"` ausgegeben wird, wenn `$alter` eine Zahl zwischen 1 und 17 enthält?
3. Wie würden Sie eine `while`-Anweisung schreiben, damit alle ungeraden Zahlen bis 49 ausgegeben werden?
4. Wie ändern Sie die `while`-Anweisung aus Frage 3 in eine `for`-Anweisung?

5.5.2 Übungen

1. Sehen Sie sich die Syntax für Kontrollstrukturen noch einmal an. Überlegen Sie, wie Kontrollstrukturen Ihre Arbeit erleichtern.
2. Sehen Sie sich den Abschnitt über den dreiteiligen Operator noch einmal an. Welchen Unterschied sehen Sie zu den nachfolgend besprochenen Kontrollstrukturen? Warum ist der dreiteilige Operator nützlich?

Kapitel 6

Funktionen

Funktionen sind das Kernstück eines guten Scripts. Durch sie wird der Code leicht lesbar und einfach zu überarbeiten. Ohne Funktionen sind große Projekte nicht zu bewältigen.

In diesem Kapitel werden Funktionen erläutert und gezeigt, wie damit repetitive Arbeiten vermieden werden.

In diesem Kapitel lernen Sie,

- wie Sie Funktionen definieren und aufrufen,
- wie Werte an Funktionen übergeben und von ihnen zurückgegeben werden,
- wie Sie Funktionen über Zeichenketten in einer Variablen dynamisch aufrufen,
- wie Sie aus einer Funktion auf globale Variablen zugreifen,
- wie Sie Funktionen mit einem »Gedächtnis« ausstatten,
- wie Sie per Referenz Daten an Funktionen übergeben.

6.1 Was ist eine Funktion?

Sie können sich eine Funktion als Maschine vorstellen. Eine Maschine verarbeitet Rohmaterial für einen bestimmten Zweck oder erstellt ein Produkt. Eine Funktion erhält Werte, verarbeitet sie und führt eine Aktion durch (zum Beispiel Ausgabe im Browserfenster) oder gibt einen neuen Wert zurück. Sie kann auch eine Aktion durchführen und einen neuen Wert zurückgeben.

Wenn Sie nur einen Kuchen backen sollen, machen Sie das wahrscheinlich selbst. Sollen es aber Hunderte oder Tausende von Kuchen sein, werden Sie nach einer Backmaschine suchen. Dasselbe gilt für Funktionen. Auch hier zählt, wie oft Sie die Aktion benötigen.

Eine Funktion ist ein in sich geschlossener Block, der durch ein Script aufgerufen und dann ausgeführt wird. Sie können Werte an Funktionen übergeben. Am Ende kann die Funktion einen Wert an den aufrufenden Block zurückgeben.

Eine *Funktion* ist ein Code, der nicht sofort ausgeführt wird. Sie wird bei Bedarf vom Script aufgerufen. Es gibt durch PHP vordefinierte Funktionen und benutzerdefinierte Funktionen. Sie erwarten häufig Eingabedaten und geben in der Regel einen Wert zurück.

6.2 Funktionsaufruf

Es gibt zwei Arten von Funktionen: eingebaute und benutzerdefinierte. Es gibt Hunderte von eingebauten PHP 4-Funktionen. Das erste Script in diesem Buch bestand nur aus einem Funktionsaufruf:

```
print("Hallo Web");
```

`print()` ist keine typische Funktion, weil hier keine Klammern nötig sind. Folgende Schreibweisen sind gültig:

```
print("Hallo Web");
```

und

```
print "Hallo Web";
```

`print()` ist eine Ausnahme. Alle anderen Funktionen brauchen Klammern, egal ob sie Argumente haben oder nicht.

Hier wurde die Funktion `print()` aufgerufen und es wurde ihr die Zeichenkette `"Hallo Web"` übergeben. Die Funktion gab die Zeichenkette im Browserfenster aus. Ein Funktionsaufruf besteht aus dem Funktionsnamen, in diesem Fall `print`, gefolgt von Klammern. Daten, die der Funktion übergeben werden sollen, werden in diese Klammern geschrieben. Die Daten in

Klammern sind Argumente. Mehrere Argumente werden durch Kommata getrennt:

```
eine_funktion( $ein_argument, $anderes_argument );
```

`print()` gibt, wie für Funktionen typisch, einen Wert zurück. Die meisten Funktionen geben am Ende eine Information zurück, und wenn es nur die Auskunft ist, ob die Durchführung erfolgreich war. `print()`gibt einen Booleschen Wert zurück.

Die Funktion `abs()`, zum Beispiel, braucht eine Zahl mit Vorzeichen und gibt den absoluten Wert dieser Zahl zurück (siehe Listing 6.1).

Listing 6.1: Aufruf der Funktion abs()

```
 1: <html>
 2: <head>
 3:   <title>Listing 6.1: Aufruf von Funktionen</title>
 4: </head>
 5: <body>
 6: <?php
 7: $zahl = -321;
 8: $neuezahl = abs($zahl);
 9: print $neuezahl;   // gibt "321" aus
10: ?>
11: </body>
12: </html>
```

Hier wird der Variablen `$zahl` der Wert `-321` zugewiesen. Die Variable wird an die Funktion `abs()` übergeben, diese macht die notwendigen Berechnungen und gibt einen neuen Wert zurück. Den neuen Wert weisen wir der Variablen `$neuezahl` zu und das Ergebnis wird im Browserfenster ausgegeben. Wir könnten auch auf temporäre Variablen verzichten, die Zahl sofort an `abs()` übergeben und das Ergebnis im Browserfenster anzeigen lassen:

```
print( abs( -321 ) );
```

Die Regeln für den Aufruf benutzerdefinierter Funktionen sind fast dieselben.

Ein *Argument* ist ein Wert, der an eine Funktion übergeben wird. Argumente werden in die Klammern des Funktionsaufrufs geschrieben. Mehrere Argumente werden durch Kommata getrennt. Die Argumente stehen der Funktion als lokale Variablen zur Verfügung.

6.3 Definition einer Funktion

Mit der Anweisung `function` definieren Sie eine Funktion:

```
function eine_funktion( $argument1, $argument2 )
     {
     // Anweisungen der Funktion
     }
```

Der Name der Funktion steht nach `function` und vor den Klammern. Wenn die Funktion Argumente braucht, werden Variablennamen in die Klammern geschrieben. Die Variablennamen werden durch Kommata getrennt. Sie stehen für Werte, die an die Funktion übergeben wurden. Auch wenn keine Argumente nötig sind, müssen die Klammern geschrieben werden. In Listing 6.2 wird eine Funktion deklariert.

Listing 6.2: Definition einer Funktion

```
 1: <html>
 2: <head>
 3:   <title>Listing 6.2: Definition eigener   Funktionen</title>
 4: </head>
 5: <body>
 6: <?php
 7: function grosshallo() {
 8:   print "<h1>HALLO!</h1>";
 9: }
10: grosshallo();
11: ?>
12: </body>
13: </html>
```

In diesem Script wird lediglich die in HTML-Tags `<H1>` eingeschlossene Zeichenkette `HALLO` im Browser ausgegeben. Wir definieren die Funktion `grosshallo()`, die keine Argumente braucht. Deshalb bleiben die Klammern leer. `grosshallo()` funktioniert zwar, ist aber nicht sehr nützlich. Die Funktion in Listing 6.3 braucht ein Argument und nutzt es sinnvoll.

Listing 6.3: Definition einer Funktion mit Argument

```
 1: <html>
 2: <head>
 3:   <title>Listing 6.3: Definition eigener Funktionen mit ⏎
      Argumenten</title>
 4: </head>
 5: <body>
 6: <?php
 7: function ausgabeBR($text) {
 8:   print "$text<br>\n";
 9: }
10: ausgabeBR("Dies ist eine Zeile");
11: ausgabeBR("Dies ist eine andere Zeile");
```

```
12: ausgabeBR("Dies ist noch eine andere Zeile");
13: ?>
14: </body>
15: </html>
```

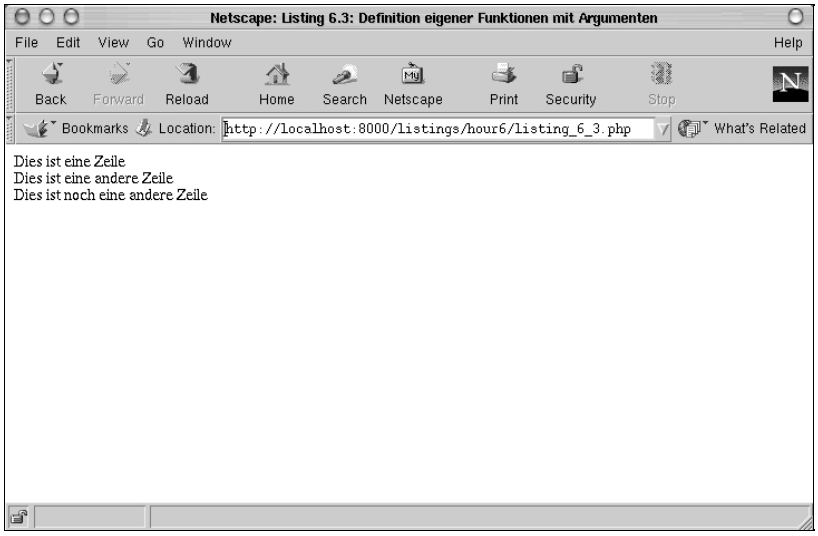


Abb. 6.1: Die Funktion gibt eine Zeichenkette mit Zeilenumbruch im Browserfenster aus.

Abbildung 6.1 zeigt die Ausgabe von Listing 6.3. Die Funktion `ausgabeBR()` erwartet eine Zeichenkette, also schreiben wir den Variablennamen `$text` in die Klammern nach der Funktionsdeklaration. Was an `ausgabeBR()` übergeben wird, wird in `$text` gespeichert. Innerhalb der Funktion setzen wir die Variable `$text` mit angefügtem `<br>`-Tag und dem Zeichen für eine neue Zeile.

Wenn jetzt eine neue Zeile im Browserfenster ausgegeben werden soll, können wir anstatt der vordefinierten Funktion `print()` die Funktion `ausgabeBR()` aufrufen. Damit ersparen wir uns für jede Zeile die Eingabe von `<br>`.

6.4 Rückgabe von Werten aus benutzerdefinierten Funktionen

Eine Funktion gibt einen Wert mit der Anweisung `return` zurück. `return` unterbricht die Funktionsausführung und gibt den Wert an den aufrufenden Code zurück. Die Funktion in Listing 6.4 gibt die Summe zweier Zahlen zurück.

Listing 6.4: Rückgabe eines Wertes

```
 1: <html>
 2: <head>
 3:   <title>Listing 6.4: Definition eigener Funktionen mit ↵
      Rückgabewert</title>
 4: </head>
 5: <body>
 6: <?php
 7: function addiereZahlen($erstezahl, $zweitezahl) {
 8:   $ergebnis = $erstezahl + $zweitezahl;
 9:   return $ergebnis;
10: }
11: print addiereZahlen(3, 5);   // gibt "8" aus
12: ?>
13: </body>
14: </html>
```

Das Script in Listing 6.4 gibt die Zahl '8' aus. Die Funktion `addiereZahlen()` sollte mit zwei numerischen Argumenten (hier 3 und 5) aufgerufen werden. Diese werden in den Variablen `$ersteZahl` und `$zweiteZahl` gespeichert. Die Funktion `addiereZahlen()` addiert die Werte dieser Variablen und speichert das Ergebnis in der Variablen `$ergebnis`. Wir können uns einen Code-Abschnitt sparen, indem wir die temporäre Variable `$ergebnis` weglassen.

```
function addiereZahlen( $ersteZahl, $zweiteZahl )
     {
     return ( $ersteZahl + $zweiteZahl );
     }
```

Die Anweisung `return` gibt einen Wert, ein Objekt oder auch nichts zurück. `return` kann einen Wert in unterschiedlicher Weise übergeben. Der Wert kann hartcodiert sein:

```
return 4;
```

Er kann Ergebnis eines Ausdrucks sein:

```
return ( $a/$b );
```

Es kann ein Wert sein, der von einem anderen Funktionsaufruf zurückgegeben wurde:

```
return ( andere_funktion( $ein_argument ) );
```

6.5 Dynamische Funktionsaufrufe

Funktionsnamen können als Zeichenketten Variablen zugewiesen werden. Die Variablen können dann wie Funktionsnamen verwendet werden. Ein Beispiel dafür sehen Sie in Listing 6.5.