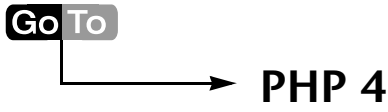

Go To
PHP 4

Die Reihe Go To

Die zuverlässigen Führer auf dem Weg zum Profi

Folgende Titel sind bereits erschienen:

Stefan Wille
Go To Java Server Pages
600 Seiten, ISBN 3-8273-1892-0

Mechtild Käufer
Go To JavaScript
624 Seiten, ISBN 3-8273-1916-1

Armin Hanisch
Go To C#
534 Seiten, ISBN 3-8273-1932-3

Drews, Kaddik, Schwab
Go To Visual Basic 6.0
768 Seiten, ISBN 3-8273-1376-7

Andreas Bohne, Guido Lang
Go To Delphi 6
960 Seiten, ISBN 3-8273-1774-6

André Willms
Go To C++-Programmierung
768 Seiten, ISBN 3-8273-1495-X

Herold, Klar, Klar
Go To Objektorientierung
744 Seiten, ISBN 3-8273-1651-0

Daryl Harms, Kenneth McDonald
Go To Python
640 Seiten, ISBN 3-8273-1800-9

Peter Loos
Go to COM
672 Seiten, ISBN 3-8273-1678-2

Guido Krüger
Go To Java 2, 2. Auflage
1.224 Seiten, ISBN 3-8273-1710-X

Dirk Abels
Go To C++ Builder 5.0
432 Seiten, ISBN 3-8273-1713-4

Michael Hernandez, John Viescas
Go To SQL
512 Seiten, ISBN 3-8273-1772-X

Pearson Education

ADDISON-WESLEY

Dirk Ammelburger

Go To PHP 4

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist
bei Der Deutschen Bibliothek erhältlich.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem
und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

05 04 03 02

ISBN 3-8273-1759-2

ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany

Einbandgestaltung: Barbara Thoben, Köln
Lektorat: Christiane Auf, cauf@pearson.de
Korrektorat: Simone Meißner, Großberghofen
Satz: reemers publishing services gmbh, Krefeld, www.reemers.de
Druck und Verarbeitung: Bercker Graphischer Betrieb, Kevelaer
Printed in Germany

Inhaltsverzeichnis

Um Ihnen die Orientierung in diesem Buch zu erleichtern, haben wir den Text in bestimmte Funktionsabschnitte gegliedert und diese durch entsprechende Symbole gekennzeichnet. Folgende Icons finden Verwendung:

Beispiele helfen Ihnen, die Erklärungen im Text schneller zu verstehen. Sie werden seitlich mit einem grauen Balken gekennzeichnet.

Manches ist von besonderer Bedeutung und verdient darum auch, besonders hervorgehoben zu werden. Manches geht ganz leicht, wenn man nur weiß, wie. Solche **Hinweise** sind sehr nützlich und bringen Sie geschwinder ans Ziel.

Es gibt in jeder Programmiersprache Fallen, in die Sie als Lernender geraten können. Die **Warnungen** im Buch führen Sie sicher an ihnen vorbei.

Mit diesem Symbol werden **Übungsfragen** am Ende vieler Kapitel gekennzeichnet, für die Sie in einem eigenen Kapitel die entsprechenden Lösungen finden.

Vorwort

Ein Vorwort zu schreiben ist gar nicht so einfach. Zum Glück ist eine Formatvorlage geduldig und schaut nicht vorwurfsvoll, wenn Buchstaben nicht so fließen wie sie sollten. Nur der Word-Assistent klopft ab und zu fragend an die Scheibe und hebt verwundert eine Augenbraue. Ganz so, als würde er sich fragen, was der sonst so eifrig tippende Mensch da draußen macht. „Ich schreibe ein Vorwort!“, wäre meine Antwort, „denk nicht weiter drüber nach ...“. Doch dazu wird es wohl nicht kommen, Computer können schließlich nur rechnen. Das tun sie so gut, dass man manchmal glauben möchte, der graue Kasten auf dem Schreibtisch sei richtig schlau.

Doch spätestens, wenn sich der Rechner wieder mal mit kritischen Ausnahmefehlern („an der Adresse 347e8df33:34d4a5f3:fff“) oder sonderbaren Meldungen („Für Glieder reserviert! Bitte Einloggung vollführen.“) meldet, holt einen die Realität wieder ein: Der PC ist nur so schlau, wie der Mensch, der ihn programmiert. Und genau darum geht es in diesem Buch: um Programmierung!

Sie werden lernen, wie Sie Ihrem Computer das „Denken“ beibringen, um praktische und intelligente Software für das Internet zu schreiben. Die Programmiersprache, die Sie für diese Aufgabe wählen, ist PHP, denn sie gehört zu den mächtigsten Skriptsprachen, die es für die Webserverprogrammierung gibt. PHP ist nicht nur schnell und einfach zu lernen, sondern auch kostenlos zu beziehen. Sie brauchen nicht eine müde Mark auszugeben, um mit dieser Sprache zu arbeiten.

Als ich vor vielen Jahren meine ersten Schritte mit der Webprogrammierung machte, habe ich mir genau so ein Buch gewünscht, wie Sie es jetzt in den Händen halten. PHP von Anfang bis Ende in allen Einzelheiten, sowie eine ausführliche Referenz aller verwendeten Befehle.

Doch bevor Sie loslegen, möchte ich noch einige wunderbare Menschen erwähnen, die mich bei der Entwicklung dieses Buches unterstützt haben: Für tiefe Einblicke in die Welt von XML und SGML danke ich Dr. Volker Deubel (dem Erfinder der Computerphilologie) von der Ludwig-Maximilan-Universität in München. Für praktische Unterstützung und den passenden Worten zu rechten Zeit gilt mein Dank meiner Lektorin Christiane Auf, die mich sicher durch dieses Buch geführt hat.

In erster Linie möchte ich mich aber bei meiner geduldigen und immer verständnisvollen Freundin bedanken, die auch nach durchschriebenen Nächten immer mit Kaffee auf mich gewartet hat.

Bevor ich jetzt aber sentimental werde, sollten Sie lieber umblättern. PHP erwartet Sie, also sollten Sie nicht zögern. Ich wünsche Ihnen viel Spaß und Erfolg bei der Lektüre dieses Buches.

Falls Sie Fragen oder Anregungen haben, können Sie mir gern schreiben: dirk.ammelburger@addison-wesley.de

Dirk Ammelburger
München, im Januar 2002

Teil I – Basics

1 Was ist PHP?

Das Internet, unendliche Weiten … Wer schon einmal gezielt nach Informationen im Netz der Netze gesucht hat, wird sich sicher schon über die Heerscharen von veralteten und darum überflüssigen Webseiten geärgert haben. In vielen Suchmaschinen stapeln sich die scheinbar einmal erstellten und danach nie wieder aktualisierten Seiten und verstopfen die Kanäle zu den wirklich aktuellen Neuigkeiten.

Eine gute Webseite steht und fällt mit ihrer Aktualität. Was gestern noch neu und aufregend war, ist heute ein alter Hut und interessiert keinen mehr. Darum ist es als Webmaster umso wichtiger, regelmäßige Updates für die eigene Seite zu erstellen, um den Surfer bei Laune zu halten. Doch wehe dem, der jetzt anfängt jeden Tag den kompletten HTML-Code seiner Seite neu zu schreiben.

1.1 Kapitelübersicht

Dieses Kapitel vermittelt einen ersten Eindruck über die Thematik. Es behandelt folgende Themen:

- Dynamisches Internet – mehr als nur HTML
- History – Wo kommt PHP her?
- Wie funktioniert PHP?
- PHP als Grundlage für Three-Tier-Anwendungen
- Vergleich mit anderen Technologien

1.2 Dynamisches Internet

Die Lösung des Problems ist eine neue Konzeption der Webseitenprogrammierung. Der HTML-Code wird nicht mehr statisch vom Webmaster der Seite erstellt, sondern generiert sich automatisch über eine serverseitige Skriptsprache, die es ermöglicht, Inhalte dynamisch zu ändern oder per Knopfdruck auszuwechseln.

Dieses Konzept bietet entscheidende Vorteile gegenüber herkömmlichen Webseiten, die auf einfachen HTML-Templates basieren. Dynamische Webseiten können einfach per Programmcode an den jeweiligen User angepasst werden und so ganz einfach seinen Wünschen und Bedürfnissen entgegenkommen. Gleichzeitig ist es möglich, über eine Datenbankschnittstelle Informationen auszulagern und unabhängig von der eigentlichen Seitenstruktur zu speichern. Man kann Informationen automatisch in eine Webseite integrieren, ohne das HTML-Design überhaupt anzurühren.

Die am häufigsten verwendete Skriptsprache zur Webserverprogrammierung ist PHP. Im Unterschied zu einfachem HTML-Code (oder JavaScript), der vom Server heruntergeladen werden muss, wird PHP-Code serverseitig ausgeführt. Das heißt, sobald ein Webbrowser eine HTTP-Anfrage an den Server stellt, wird das Skript geladen und durch einen Interpreter ausgeführt. Das Ergebnis dieses Vorgangs ist in der Regel ein reiner HTML-Code, der dann an den Server geschickt wird. Der User bekommt den Quellcode des Skripts nicht zu sehen (was z.B. bei JavaScript immer der Fall ist).

PHP ähnelt in der Syntax stark den Sprachen C oder auch Perl. Damit ergeben sich auch zwangsläufig Ähnlichkeiten mit vielen C-ähnlichen Sprachen wie Java oder C#. Wenn Sie bereits Kenntnisse in einer dieser Sprachen haben, wird Ihnen der Einstieg sicher leicht fallen.

1.3 History

PHP ist eine relativ junge Sprache im Vergleich zu anderen Programmiersprachen. Sie wurde speziell für den Einsatz auf Webservern zur Generierung von Internetseiten entwickelt. Der Vater dieser Sprache ist Rasmus Lerdorf, der Anfang 1995 die Grundlagen für PHP geschaffen hat. Die damalige Abkürzung stand für die simple Bezeichnung *Personal Homepage Tool*. Inzwischen hat sich die Interpretation des bekannten Kürzels gewandelt, und so steht PHP heute für die Bezeichnung *Hypertext Preprozessor.*

PHP ist inzwischen in der Version 4 im Einsatz und hat seit der Urversion eine gewaltige Entwicklung hinter sich gebracht. Vor allem der Sprung von der Version 3 zur aktuellen Version 4 hat komplette interne Neustrukturierungen nach sich gezogen. War PHP3 noch eine reine Interpretersprache, die für die Ausführung des Codes Schritt für Schritt durch das Programm ging, so ist die aktuelle Version 4 als Bytecode-Compiler ver-

fügbar. Die intern verwendete ZEND-Engine kompiliert den Code vor der Ausführung und erreicht so eine erheblich verbesserte Verarbeitungsgeschwindigkeit gegenüber PHP3.

Ein weiterer Unterschied zwischen PHP3 und PHP4 ist die Einbindung der objektorientierten Programmierung (OOP), die es erlaubt, Programme in Klassenstrukturen aufzuteilen. Außerdem bietet PHP4 jetzt eine automatische Session-Verwaltung und verbesserte Array-Funktionen.

Neuerungen von PHP4

- automatische Freigabe von Speicher
- boolesche Konstanten
- OOP-Unterstützung
- `foreach()`-Schleifen
- Pointer
- Sessions
- Entfernen von Variablen aus dem Speicher per `unset()`
- schnellere Verknüpfung von Strings und Variablen
- einfache Webserveranpassung

Dieses Buch beschäftigt sich ausschließlich mit PHP4 und dessen Möglichkeiten. Viele der besprochenen Punkte sind abwärtskompatibel, nichtsdestotrotz empfehle ich, nur mit der aktuellen Version zu arbeiten. Da die meisten Webserver im Netz die neueste Version inzwischen installiert haben sollten, dürfte das auch kein Problem sein.

1.4 Wie funktioniert PHP?

PHP ist im Gegensatz zu anderen Server-Skriptsprachen wie Perl ausschließlich für den Einsatz im Web entwickelt worden. Das wird vor allem beim praktischen Einsatz der Sprache deutlich, wo das Programm nicht wie bei Perl in einem eigenen Scriptfile stehen muss, sondern einfach in den HTML-Code eingebunden wird. Der Vergleich zu JavaScript, das ebenfalls direkt in dem HTML-Code eingebunden wird, ist nahe liegend und durchaus berechtigt, da der Vorgang bei beiden Sprachen ähnlich ist. Der große Unterschied liegt allerdings im Detail: JavaScript wird vom Browser ausgeführt, PHP vom Interpreter auf dem Server.

Genauer betrachtet bedeutet dies, dass der JavaScript-Code zuerst vom Server auf den Client-Rechner transferiert wird und dann erst vom Browser ausgeführt werden kann. Dieses Vorgehen ist nicht immer vorteilhaft, denn der Programmierer weiß vorher nicht, auf welchem System sein Code landen wird. So kann es passieren, dass der Browser überhaupt kein JavaScript unterstützt und die Webseite nicht funktioniert. Das zweite große Manko ist die Sichtbarkeit des Skripts. Jeder User kann den Code der Webseite einsehen und damit die Funktionalität des Skripts nachvollziehen. Sicherheitsrelevante Aspekte sind mit JavaScript also nicht zu lösen.

Im Gegensatz dazu ist PHP eine serverseitige Skriptsprache. Eine Anfrage von einem HTTP-Client (Browser) wird nicht mit dem sturen Absenden der geforderten Seite quittiert, sondern zieht eine Reihe von Aktionen auf dem Server nach sich. Anhand der Endung der geforderten Datei (in der Regel .php oder .php4) erkennt der Server das PHP-Skript. Daraufhin startet er den PHP-Interpreter, der das angeforderte Dokument übersetzt und den Code ausführt. Die Befehle und Funktionen im Skript werden interpretiert und durch den entsprechenden HTML-Code ersetzt, der an derselben Stelle in Dokument erscheint. Nach der Übersetzung des Scripts wird das modifizierte Ergebnis an den Client-Rechner geschickt, der nur den HTML-Code der Ausgabe zu sehen bekommt.

Auf dem Server selbst erfolgt die Kommunikation über die Serversoftware, die den Datenstrom der verschiedenen Programme koordiniert. Innerhalb der Serversoftware (in den meisten Fällen Apache) wird die PHP-Software als Modul geladen, so dass diese immer im Hintergrund läuft. In älteren Versionen wurde PHP auch über das Common Gateway Interface (CGI) angesprochen, was aber bedeutend langsamer in der Ausführung war.

Über das PHP-Modul werden dann die geforderten Skripte geladen und mit den nötigen Daten (Datenbank) verknüpft. Das fertige Ergebnis wird über den Server an den Client weitergeleitet, der das Ergebnis in fertigem HTML betrachten kann.

Dem User bleibt das Skript der Seite also verborgen und er sieht nur den jeweiligen HTML-Code. Die eigentliche Webseite wird also erst auf Anfrage dynamisch generiert und kann so in Abhängigkeit von den Benutzeraktionen entsprechend modifiziert werden. Der Nachteil einer dynamischen Seite ist die langsamere Zugriffszeit für den Surfer. Da jeder Seitenaufruf eine neue Interpretation des Skriptcodes nach sich zieht, ist eine PHP-Seite etwas langsamer als eine statische HTML-Seite.

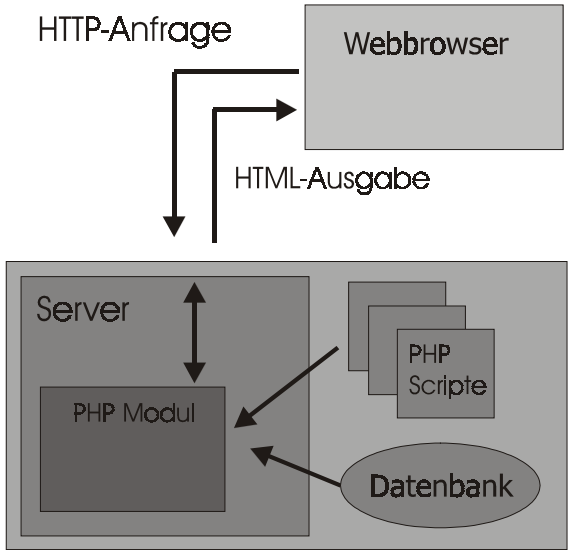


Abbildung 1.1: PHP4 in der Anwendung

1.5 PHP als Grundlage für Three-Tier-Anwendungen

Eine moderne Webseite, die Informationen publiziert oder Daten verwalten muss, braucht eine klare Struktur in ihrem Aufbau. Grundsätzlich unterscheidet man zwischen drei verschiedenen Ebenen in der Programmierung, die klar voneinander getrennt werden sollten. Daraus ist das Prinzip der Three-Tier-Anwendungen (Drei Schichten) entstanden, die eigentlich für jedes Programm als Grundlage dienen sollte. Wir teilen also ein Programm wie folgt auf:

Die erste Ebene ist die Anwendungsebene, die als Schnittstelle zum User fungiert. Hier werden alle Programmausgaben realisiert und die Eingaben vom Benutzer entgegengenommen. Diese Schnittstelle sollte möglichst komfortabel gestaltet sein und relativ unabhängig vom Rest des Programms stehen können. Da wir uns mit der Programmierung für Internetuser beschäftigen, ist die Anwendungsebene in der Regel eine Webseite, die der Browser darstellt.

Die zweite Ebene ist das eigentliche Programm, das die Funktionalität der Anwendung gewährleistet. Diese Programmebene nimmt die Eingaben der Anwendungsebene auf und setzt sie um. Gleichzeitig gibt man hier alle

Ergebnisse an den User weiter, die er dann wieder über seine Schnittstelle der Anwendungsebene zu sehen bekommt. In diesem Buch werden wir die Funktionalität unserer Programme mit PHP realisieren. Diese Skripte sind die Programmebene für unsere Three-Tier-Anwendung.

Die dritte und letzte Ebene ist der Bereich für die Datenverwaltung. Hier werden die notwendigen Informationen für die Anwendung gespeichert und bei Bedarf abgerufen oder aktualisiert. Diese Schicht wird je nach Aufwand über eine Datenbank oder einfache Textdateien auf dem Server realisiert.

Selbstverständlich ist es möglich, die verschiedenen Schichten miteinander zu kombinieren oder einfach zusammenzufassen, aber im Zweifelsfall empfiehlt es sich, diese strikte Trennung einzuhalten. So sollten Daten möglichst nie in HTML-Code zwischengespeichert werden, ein PHP-Skript nie mit Konstanten überflutet werden.

Die Trennung des Interface vom eigentlichen Programm sollte bei der Webprogrammierung auch so verstanden werden, dass das eigentliche Layout vom Skript separiert werden sollte. Es bietet sich an, den HTML-Quellcode der Seiten getrennt anzufertigen und dann über so genannte Templates im Programm unterzubringen.

1.6 Vergleich mit anderen Technologien

Es gibt eine Reihe verschiedener Möglichkeiten Webseiten dynamisch zu gestalten, die alle ihre Vor- und Nachteile haben. Eine gute Möglichkeit, eine Sprache zu charakterisieren, ist den Vergleich zu einer ähnlichen Sprache zu suchen.

1.6.1 Vergleich mit Perl/CGI

Einer der nahe liegendsten Vergleiche ist die Skriptsprache Perl, die sozusagen die Mutter aller dynamischen Webseiten ist. Als anfängliche reine UNIX-Sprache wurde Perl in der Mitte der fünfziger Jahre zur Systempflege und Verwaltung für Systemadministratoren geschaffen. Im Laufe der Zeit ist Perl über diese Aufgaben hinausgewachsen und war mit der Einführung des HTTP-Protokolls eine der ersten Möglichkeiten über das CGI-Interface Daten zu bearbeiten.

Perl ist eine sehr mächtige Sprache und verfügt über eine Reihe von Möglichkeiten (über die Webprogrammierung hinaus), die PHP nicht zu bieten hat. Allerdings macht das die Sache auch komplizierter als eigentlich nötig, und so ist Perl inzwischen nur noch die zweite Wahl.

Der zweite Punkt, der gegen Perl spricht, ist das inzwischen veraltete Common Gateway Interface. Für jede Anfrage an den Server muss das jeweilige Programm neu gestartet werden. Das kostet Zeit und macht die Webseite langsam.

1.6.2 Vergleich mit ASP

Auf den ersten Blick haben PHP und ASP (Active Server Pages) einige Gemeinsamkeiten, die daher rühren mögen, dass PHP Teile von ASP übernommen hat. Beide Sprachen erlauben den Programm-Code in die HTML-Seite einzubetten. Die zweite Gemeinsamkeit ist die einfache Handhabung der Sprache, die einfach auf dem Server einzurichten ist.

Der große Unterschied zwischen ASP und PHP ist gleichzeitig auch der große Vorteil von PHP: Da ASP von Microsoft entwickelt wurde, läuft der Interpreter für diese Skriptsprache nur auf den Microsoft-eigenen Servern IIS (Internet Information Server) und dem Personal Web Server. ASP fehlt also jede Plattformunabhängigkeit, die PHP so flexibel macht.

Ein weiterer Punkt ist die unterschiedliche Syntax zwischen ASP und PHP. Während PHP auf der Syntax von C oder Java beruht, benutzt Microsoft eine Abart der »Haussprachen« Vbscript oder JScript.

1.6.3 Vergleich mit JSP und Servlets

Servlets und Java Server Pages funktionieren ähnlich wie ein PHP-Skript, mit dem Unterschied, dass diese komplett in Java geschrieben werden. Da Java in seiner Komplexität ein ganzes Stück schwerer zu erlernen ist als eine Skriptsprache wie PHP, stellt sich die Entwicklung von javabasierten Webseiten ein Stück komplizierter dar. Außerdem ist die Verbreitung von Servern, die JSP und Servlets unterstützen, nicht so groß wie die von PHP-Servern.

Der Vorteil von Java ist die höhere Arbeitsgeschwindigkeit, da der Code nicht jedes Mal aufs Neue kompiliert werden muss.

2 Schnelleinstieg

In diesem Kapitel lernen Sie die Grundsätze von PHP kennen und erfahren, welche Voraussetzungen und Werkzeuge Sie benötigen. Sie erhalten einen Überblick über die Arbeitsweise mit PHP. Außerdem wird die Struktur des Buches besprochen und erklärt, wie Sie am besten damit arbeiten können.

2.1 Kapitelübersicht

- Das Arbeitsumfeld
- Integrated Development Environment
- Der Editor
- Für wen ist das Buch?
- Wie sollten Sie das Buch lesen?

2.2 Das Arbeitsumfeld

Bevor Sie loslegen und das erste Programm in PHP schreiben, müssen Sie sich noch über ein paar Punkte klar werden. Ein PHP-Programm ist ein Stück Software, das für einen Netzwerkrechner (Server) geschrieben wird. Das heißt, dass andere Rechner im Internet oder ein beliebiges Intranet auf den Server zugreifen und so die Skripte aktivieren.

Wenn Sie bereits andere Programmiersprachen kennen gelernt haben, werden Sie es vielleicht gewöhnt sein, ein Programm auf Ihrem Rechner zu schreiben und im selben Atemzug zu kompilieren und zu testen. Das funktioniert bei Applikationen, die für einen Anwenderrechner geschrieben wurden, recht gut. Selbst statische Webseiten mit JavaScript-Elementen oder Java-Applets können (wenn auch eingeschränkt) lokal getestet werden.

Für Webserverprogrammierung (speziell für PHP-Skripte) gilt das nicht, da auf einem lokalen Rechner eine völlig andere Umgebung herrscht als auf einem Netzwerkserver. Der einfachste und logischste Schritt wäre ein fertiges Skript per FTP auf einen PHP-fähigen Server zu kopieren und dort zu testen. Für den Anfang ist dieses Vorgehen durchaus empfehlenswert, auch wenn es manchmal etwas umständlich scheint.

Ein fertig eingerichteter Webserver hat den Vorteil eines gemachten Nestes, in das man sich nur noch zu setzen braucht. Ein fleißiger Systemadministrator hat uns dort schon alle Arbeiten abgenommen und sowohl PHP als auch einen Webserver mit Datenbank für uns eingerichtet.

Für die Entwicklung größerer Projekte empfiehlt es sich, eine eigene Entwicklungsumgebung auf dem Rechner zu schaffen, um die Programme auch lokal testen zu können. Alle Beispiele in diesem Buch wurden für die Zusammenarbeit mit den drei folgenden Softwarepaketen geschrieben:

- Apache Webserver (*www.apache.org*)
- PHP4 (*www.zend.com*)
- Datenbankserver MySQL (*www.mysql.com*)

Alle drei Pakete können kostenlos aus dem Internet heruntergeladen werden. Die jeweils aktuellen Versionen sind im Internet unter den angegebenen Webadressen zu finden.

Allen Paketen ist eine ausführliche Installationsanleitung beigelegt. Falls es dennoch Probleme geben sollte, findet sich im zweiten Teil des Buches ein eigenes Kapitel über die Installation und Einrichtung der Software.

Die im Folgenden vorgestellten Skripte sind auf der CD-ROM des Buches abgelegt und können ebenfalls auf der Webseite *www.kulturbrand.de* heruntergeladen werden.

2.3 Integrated Development Environment

Wenn Sie eine Entwicklungsumgebung (IDE) benutzen wollen, um Ihre Programme zu schreiben, dann gibt es eine umfangreiche Auswahl von Free- und Shareware im Internet, die einem das Leben erleichtert. Es empfiehlt sich, ein Programm mit Syntax-Highlighting zu wählen, weil es Programmcode bedeutend übersichtlicher macht.

Einige Programme bieten sogar eine Korrektur des integrierten HTML-Codes an oder erlauben, den Code lokal zu debuggen, um eventuelle Fehler zu finden. Die beiliegende CD-ROM bietet ein kleine Auswahl von Programmen, die sich als sehr nützlich erwiesen haben.

2.4 Der Editor

Falls Sie auf eine Entwicklungsumgebung verzichten wollen, können Sie jedes PHP-Programm in jedem einfachen Editor schreiben. Ob Sie nun Notepad von Windows verwenden oder ganz klassisch vi von UNIX spielt keine Rolle, wichtig ist, dass das Programm keine Formatierungsanweisungen im Sourcecode hinterlässt. Textverarbeitungsprogramme wie Word oder Starwrite sind gänzlich ungeeignet.

Übrigens: Alle hier vorgestellten Programme wurden mit Notepad erstellt.

2.5 Für wen ist das Buch?

Das Buch richtet sich an alle Leser, die schon Erfahrung im Umgang mit dem PC und dem Internet haben. Sie sollten schon einmal eine Webseite erstellt haben und den Umgang mit HTML, FTP und einem Webbrowser gewohnt sein.

Für das Verständnis des Buches brauchen Sie keine Programmiererfahrung. Alle notwendigen Grundlagen werden in den ersten Kapiteln geklärt, die inhaltlich aufeinander aufbauen.

Im Bereich der HTML-Programmierung wäre es gut, wenn Sie schon einmal mit Formularen gearbeitet hätten und wüssten, wie sich eine URL zusammensetzt. Weiterhin wäre es sehr hilfreich, wenn Sie bereits eine grundlegende Vorstellung von den Protokollen im Internet hätten, auch wenn dies keine wirkliche Voraussetzung ist.

Prinzipiell kann man sagen, dass sich dieses Buch in seinem Aufbau an verschiedene Lesergruppen gleichzeitig wendet, da es sowohl als Einführungslektüre als auch als Nachschlagewerk geeignet ist.

Wenn Sie ein Buch gerne von hinten nach vorne durchlesen, haben Sie das richtige Werk in der Hand, da der Kapitelaufbau ein schnelles und effektives Lernen erlaubt. Gleichzeitig ist es aber auch möglich, im zweiten Teil des Buches, Going Professional!, einfach die wesentlichen Kapitel aufzuschlagen und ohne Probleme in die Materie einzusteigen. Leser, die ein ganz bestimmtes Ziel verfolgen, sind mit dieser Technik gut beraten.

2.6 Wie sollten Sie dieses Buch lesen?

Dieses Buch ist in drei Teile aufgeteilt, die Sie prinzipiell unabhängig voneinander lesen können. Diese Struktur erleichtert so den Einstieg in die jeweilige Materie.

Der erste Teil, Basics, behandelt die Grundlagen von PHP und schafft einen Überblick über die Programmiersprache. Alle, die schon Erfahrungen in der Webprogrammierung haben, können hier ihr Wissen auffrischen und etwas wiederholen. Alle anderen haben in diesen Kapiteln die Möglichkeit, den Einstieg in die Sprache PHP zu schaffen, ohne sich dabei in Einzelheiten zu verlieren. Es werden kompakt und zügig die Grundlagen geschaffen, die für den Rest des Buches nötig sind. Gleichzeitig soll dieses Kapitel auch Lust auf mehr machen, denn es gibt an vielen Stellen einen Ausblick auf die späteren Kapitel, in denen es dann wirklich interessant wird.

Teil zwei des Buches, GoingPrefessional!, befasst sich mit den fortgeschrittenen Themen, wie Datenbankanbindungen, objektorientierte Programmierung oder Sessionverwaltung. Gleichzeitig gebe ich am Rande immer wieder Tipps und Anregungen, wie ein Programm besser läuft oder Zeit gespart werden kann. Ein komplettes Kapitel wird sich mit der Abfragesprache SQL befassen, die die Voraussetzung für die Arbeit mit Datenbanken ist. Nebenher werde ich Ihnen auch etwas über XML und die dynamischen Grafiken erzählen, denn PHP ist weitaus mächtiger, als es oft präsentiert wird.

Der dritte und letzte Teil des Buches, Reference, enthält die Referenz der PHP-Syntax sowie einen Überblick über die wichtigsten SQL-Befehle. Einige Listen mit verschiedenen Konstanten und Servervariablen runden das Angebot ab. Hinter der Referenz finden Sie Angaben zu weiterführender Literatur.

Ich empfehlen Ihnen als Einführung auf jeden Fall den ersten Teil des Buches zu lesen. Danach haben Sie die Möglichkeit, nach Belieben in den Kapiteln zu springen und sich die Kapitel rauszusuchen, die für Sie am interessantesten sind.

Im Prinzip ist jedes Kapitel im zweiten Teil des Buches völlig unabhängig zu lesen und kann ohne Vorwissen aus den vorherigen Seiten gelesen werden. An vielen Stellen verweise ich allerdings auf andere Kapitel, die mit dem aktuellen Stoff in Zusammenhang stehen. So wird ein kleines Netzwerk durch das ganze Buch entstehen, das auch schnelle Sprünge erlaubt, ohne Wissenslücken zu hinterlassen.

3 Grundlegende Sprachelemente von PHP

Die Syntax von PHP basiert zu großen Teilen auf der Programmiersprache C. Außerdem wurden einige Eigenschaften aus den Sprachen Java und Perl übernommen. Die Struktur jedes PHP-Programms besteht aus einer Folge von Funktionen und Anweisungen.

3.1 Kapitelübersicht

- Die Syntax von PHP
- Operatoren
- Datentypen
- Variablen
- Strings
- Arrays

3.2 Die Syntax von PHP

Der PHP-Code wird direkt in den HTML-Code eingebettet und kann dort an beliebigen Stellen auftreten. Es ist möglich, dass der Code an mehreren Stellen durch HTML unterbrochen wird. Beginn und Ende des Codes werden durch spezielle Tags gekennzeichnet, die dem Interpreter signalisieren, wo seine Arbeit beginnt und wo sie endet. Es gibt verschiedene Möglichkeiten diese Tags zu gestalten.

Eine der am häufigsten verwendeten Möglichkeiten ist der so genannte SGML (Standard Generalized Markup Language)-Stil. Meiner Meinung nach ist dies auch die einfachste Möglichkeit PHP-Code einzubinden, da die Tags am kürzesten sind und den Quellcode übersichtlich gestalten.

```
<?
echo "Der SGML-Stil";
?>
```

Das Skript wird einfach von zwei eckigen Klammern mit einem Fragezeichen umschlossen. Ich werde alle Programme auf diese Weise kennzeichnen. Der Vollständigkeit halber stelle ich noch die weiteren Möglichkeiten vor.

Eine weitere Möglichkeit der Einbindung ist der XML (Extensible Markup Language)-Stil, der im Prinzip genauso aufgebaut ist wie unser erstes Beispiel, aber die Sprachdefinition genauer festlegt.

```
<?PHP
echo "Der XML-Stil";
?>
```

Es ist möglich, die Kennzeichnung mit den Buchstaben PHP sowohl in Groß- als auch in Kleinbuchstaben vorzunehmen.

Den ASP (Active Server Pages)-Stil können Sie nur verwenden, wenn die PHP-Konfiguration diese Einbindung vorsieht. In der Regel ist dies nicht der Fall, aber wahrscheinlich wird Ihnen bei dieser Frage Ihr Systemadministrator weiterhelfen können.

```
<%
echo "Der ASP-Stil";
%>
```

Ein sehr elegante Methode, den PHP-Code einzubinden, ist über den (aus JavaScript bekannten) <SCRIPT>-Tag möglich.

```
<SCRIPT language="php">
echo "Der JavaScript-Stil";
</script>
```

Allerdings bedeutet dies einen hohen Schreibaufwand, den die meisten Programmierer scheuen.

3.2.1 Kommentare

Kommentare ermöglichen es, Hinweise und Bemerkungen im Sourcecode zu hinterlassen, die helfen, das Programm zu verstehen. Gleichzeitig können auf diese Weise auch Urhebervermerke gemacht werden.

Ein Kommentar beginnt in PHP mit den Zeichen /* und endet mit */. Diese Kommentare umschließen einen gewissen Bereich im Sourcecode und können sich auch über mehrere Zeilen erstrecken.

```
/*Dies ist ein langer Kommentar,
der sich über mehrere Zeilen
erstreckt! */
```

Außerdem kennt PHP noch die Möglichkeit von einzeiligen Kommentaren, die aus C übernommen wurden. Jeder Kommentar beginnt mit den Zeichen // . Diese Art von Kommentar bezieht sich lediglich auf die aktuelle Zeile und eignet sich nur für kurze Bemerkungen.

```
$str = "Hallo Welt!"    //Dies ist ein Kommentar
echo $str;    //Dies auch!
```

Der Doppelslash wird gerne dazu verwendet, Zeilen aus dem Code direkt zu kommentieren, ohne dabei die Zeile zu wechseln.

3.2.2 Anweisungen

Ein PHP-Skript besteht aus einer Reihe von Anweisungen, die vom Interpreter der Reihe nach abgearbeitet werden. Eine Anweisung ändert in der Regel Daten im Programm oder interagiert mit der Programmumgebung. Die Syntax einer Anweisung ist an C angelehnt und endet immer mit einem Semikolon.

Mit diesen Voraussetzungen können wir jetzt unser erstes kleines (in HTML eingebettetes) Programm schreiben, das die Anweisung *echo* benutzt. Dieser Befehl erlaubt es, Zeichenketten an den Browser auszugeben.

```
<html>
<head>
<title>PHP Programm</title>
</head>
<body>
<?      //Hier beginnt der PHP-Code!
echo "Hallo PHP!";
?>      //Hier endet der PHP-Code!
</body>
</html>
```

Wenn Sie dieses Programm auf einem PHP-fähigen Server laden und über einen Browser abrufen, erhalten Sie folgende Ausgabe:

```
Hallo PHP!
```

Durch das HTML-Konstrukt ist die Ausgabe als Webseite gekennzeichnet worden. Das wäre allerdings nicht wirklich nötig gewesen, da jede PHP-Seite automatisch als text/html gekennzeichnet ist.

3.2.3 Ausdrücke

PHP ist eine ausdrucksorientierte Sprache. Das heißt, dass sich alle Anweisungen und Daten über bestimmte Syntaxformen definieren und deshalb einem Reglement unterliegen. Ein wesentlicher Punkt ist, dass ein Ausdruck immer einen Wert haben muss, der je nach Datentyp standardmäßig entweder 0 oder »« (also ein Leerstring) ist.

Die wichtigsten Fragen, die sich in jedem Scripting stellen, sind: Welchen Wert hat welcher Ausdruck? Welcher Wert wird einem Ausdruck zugewiesen? Wie hat sich der Ausdruck verändert?

Das klingt alles ein wenig abstrakt für den Anfang, aber das wird sich sofort ändern. Ein elementarer Ausdruck in der gültigen PHP-Syntax könnte so aussehen:

```
20
```

PHP erkennt diesen Ausdruck als Integer-Konstante mit dem Wert 20. Allerdings ist dieser Ausdruck für sich allein gestellt nicht sehr sinnvoll, weil PHP damit keine Aktion verbindet. Nur im Zusammenhang mit einer Anweisung oder einer Zuweisung kann PHP einen Ausdruck nutzen.

Ein weiteres Beispiel für Ausdrücke sind Variablen, die im Gegensatz zu einer Konstanten ihren Wert ändern können. Diese Eigenschaft macht sie zu Datenspeichern in einem Programm und nutzbar für den Transport von Informationen. Eine Variable wird in PHP mit einen Dollarzeichen gekennzeichnet.

Eine Variable für sich allein gestellt ist genauso sinnlos wie eine konstante Zahl. Die einfachste Form einer Anweisung ist deshalb eine Zuweisung, die es erlaubt, Werte von einem Ausdruck in einem anderen zu transferieren.

```
$betrag = 100;
```

Diese Anweisung ist ein zusammengesetzter Ausdruck und setzt den Wert der Variablen $betrag auf den Wert 100. PHP arbeitet sich in solchen

Situation immer von rechts nach links. Der rechte Wert weist dem Wert auf der linken Seite des Gleichheitszeichens (Zuweisungsoperator) den Wert zu.

Genauso ist es möglich, Werte einer Variablen einer anderen zuzuweisen.

```
$betrag = $summe;
```

$betrag hat jetzt denselben Wert wie $summe, der aus der vorangegangenen Zuweisung 100 ist. Würden wir das vorherige Skript ignorieren, dann wäre die folgende Anweisung gleichbedeutend:

```
$betrag = $summe = 100;
```

Beide Variablen haben jetzt den Wert 100.

3.2.4 Funktionale Ausdrücke

Nichtelementare Ausdrücke sind beispielsweise Funktionen. Sie haben in einem Skript immer den Wert ihres Rückgabewertes und können so zugewiesen werden.

```
function wert()
{
   return 20;
}
$x = wert();
```

Die Variable $x hat jetzt den Wert 20, der von der Funktion `wert()` zurückgegeben wurde. Fehlt einer Funktion ein Rückgabewert, dann wird $x auf einen Standardwert (hier 0) gesetzt.

3.2.5 TRUE und FALSE

Wahr und falsch ist nicht nur in der Moral ein wesentlicher Aspekt, sondern auch in der Programmierung. Im Gegensatz zu PHP3 hat PHP4 die Möglichkeit zwischen den booleschen Konstanten TRUE (wahr) und FALSE (falsch) zu unterscheiden.

Ein Ausdruck ist dann wahr, wenn ihm die Kostante TRUE zugewiesen wurde. Umgekehrt gilt ein Ausdruck als falsch, wenn ihm der Wert FALSE zugewiesen wurde.

```
<?
$bool = TRUE;
$bool2 = FALSE;
echo $bool;
echo "<br>";// Zeilenumbruch
echo $bool2;
?>
```

Ausgabe:

```
1
0
```

In der Version 3 von PHP gibt es diese Konstanten TRUE und FALSE nicht, die intern als die Integerwerte 1 und 0 behandelt werden. Der Wahrheitswert eines Ausdrucks wird auf andere Weise ermittelt, die in PHP4 genauso gültig ist. Jeder positive oder negative nummerische Wert, der nicht 0 ist, sowie jede nichtleere Zeichenkette hat den Wert TRUE. Die 0 und die leere Zeichenkette »« wird als FALSE gewertet. Arrays werden als FALSE betrachtet, wenn sie keine Elemente enthalten.

3.2.6 Zuweisung »by value«

Wenn PHP einer Variable einen Wert zuweist, dann geschieht das über eine Zuweisung *by value*, also direkt über die Wertigkeit der Zuweisung. Der Wert der einen Variable wird ausgelesen und in die zweite Variable geschrieben. Das bedeutet also, dass beide Ausdrücke (Variablen) unabhängig voneinander existieren und jede ihren eigenen Platz im Arbeitsspeicher belegt.

Nach der Zuweisung haben beide Variablen zwar denselben Wert, aber ansonsten keine Gemeinsamkeit mehr. Wird eine Variable nachträglich geändert, hat dies keine Auswirkungen auf die andere Variable.

```
<?
$x = 20;
$y = $x;
$x = 40;
echo $x;
echo "<br>";    // Zeilenumbruch
echo $y;
?>
```

Listing 3.1: Call by value

Ausgabe:

```
40
20
```

3.2.7 Zuweisung »by reference«

PHP3 kannte nur die Zuweisung *by value*, wie wir sie im letzten Kapitel kennen gelernt haben. PHP4 dagegen kennt eine weitere Form der Zuweisung, die erfahrenen Programmierern vielleicht als Pointer oder Referenzen bekannt ist. Man spricht auch von einer Zuweisung *by reference*, die im Gegensatz zur Zuweisung *by value* nicht einen Wert einer Variablen zuweist, sondern das Ziel auf den Wert einer Speicheradresse setzt.

Das Ergebnis ist, dass wir zwei unterschiedliche Variablen haben, die auf dieselbe Adresse im Speicher verweisen und den dort gespeicherten Wert zurückgeben. Der Unterschied zur Zuweisung *by value* äußert sich darin, dass bei einer Veränderung der einen Variablen die andere ebenfalls verändert wird.

```
<?
$x = 20;
$y = &$x;       //Verweis auf dieselbe Speicherstelle
$x = 40;
echo $x;
echo "<br>";    // Zeilenumbruch
echo $y;
?>
```

Listing 3.2: Call by reference

Ausgabe:

```
40
40
```

Sowohl $x als auch $y geben jetzt den Wert 40 aus. Das beweist, dass es sich im Prinzip um dieselbe Variable handelt.

3.2.8 Übungen

1. Welchen Sinn haben Kommentare? Auf welche Weise werden sie in PHP realisiert?
2. Was sind Anweisungen? Wie werden Anweisungen in PHP markiert?
3. Erläutern Sie die Konstanten TRUE und FALSE.
4. Erläutern Sie den Unterschied zwischen einer Zuweisung *by value* und einer Zuweisung *by reference*. Wie sieht dieser Unterschied syntaktisch aus?

3.3 Operatoren

In einem der letzten Kapitel haben Sie gesehen, wie man verschiedene Ausdrücke über einen Zuweisungsoperator (=) manipulieren kann. PHP bietet eine ganze Reihe von unterschiedlichen Operatoren an, die einem die Programmierarbeit erleichtern. Die meisten Operatoren werden Ihnen aus anderen Sprachen wie C oder sogar Basic bekannt sein, auch wenn PHP weit mehr zu bieten hat.

3.3.1 Arithmetische Operatoren

Ganz am Anfang stehen natürlich die einfachen mathematischen Operatoren für die vier Grundrechenarten sowie der Modulo-Operator, der den ganzzahligen Rest einer Division zurückgibt.

Name	Beispiel	Beschreibung
Addition	$a + $b	Summe der Werte
Subtraktion	$a – $b	Differenz der Werte
Multiplikation	$a * $b	Produkt der Werte
Division	$a / $b	Quotient der Werte
Modulo	$ % $b	Rest der Ganzzahlendivision

Tabelle 3.1: Arithmetische Operatoren

```
<?
$erg = 5 + 3;
echo $erg;
echo "<br>";    //Zeilenumbruch!
```

```
$erg = 5 - 3;
echo $erg;
echo "<br>";    //Zeilenumbruch!
$erg = 5 * 3;
echo $erg;
echo "<br>";    //Zeilenumbruch!
$erg = 5 / 3;
echo $erg;
echo "<br>";    //Zeilenumbruch!
$erg = 5 % 3;
echo $erg;
echo "<br>";    //Zeilenumbruch!
?>
```

Listing 3.3: Arithmetische Operatoren

Ausgabe:

```
8
2
15
1.6666666666667
2
```

Zu diesen Operatoren braucht man eigentlich nicht viel zu sagen, da sie intuitiv verwendet werden können. Wem die Modulo-Rechnung unter diesem Namen unbekannt ist, der kennt sie vielleicht als Division mit Rest oder schlicht »Rest-Rechnung«. Es geht darum, den Anteil aus dem Zähler der Rechnung zu ziehen, der nicht mehr glatt durch den Nenner geteilt werden kann.

Diese Rechnung ist sehr praktisch für die Bestimmung von Intervallen in einer Zählschleife.

3.3.2 Zuweisungsoperatoren

PHP kennt nur einen zentralen Zuweisungsoperator, den wir in den letzten Kapiteln schon ausgiebig genutzt haben. Über das Gleichheitszeichen kann ein Ausdruck einem anderen Ausdruck zugewiesen werden.

```
$a = 100;
$b = $a;
$c = $d = 4 * 25;
```

Alle diese Zuweisungen sind gültig und haben denselben Effekt. Die Variablen werden auf den Wert 100 gesetzt.

Neben dem elementaren Zuweisungsoperator stellt PHP eine Reihe von Operatoren zur Verfügung, die ebenfalls Zuweisungsaufgaben übernehmen. Es handelt sich hierbei immer um Abkürzungen der »offiziellen« Syntax, die dem Programmierer das Tippen erleichtert. Die meisten Operatoren sind aus dem Sprachgebrauch von C entnommen.

Inkrementierung

Mithilfe des Inkrementierungsoperators kann der Wert einer Variable inkrementiert, dass heißt um den Wert 1 erhöht werden. Der normale Weg, dies zu tun, würde in PHP-Syntax so aussehen:

```
$x = x + 1;
```

Das ist zwar korrekt, aber nicht sehr schnell zu schreiben. Da diese Konstruktion sehr oft in vielen Programmen auftaucht, wurde eine Abkürzung geschaffen, die genau dasselbe bewirkt.

```
$x + +;
```

Dieser Ausdruck erhöht den Wert der Variablen $x um 1. PHP unterscheidet zwischen zwei verschiedenen Arten der Inkrementierung. Die erste Version haben wir gerade kennen gelernt. Die zweite Version sieht so aus:

```
+ + $x;
```

Auch hier wird $x um 1 erhöht. Der Unterschied liegt im Detail: Bei der ersten Version wird zuerst der Wert der betroffenen Variable zurückgegeben und erst dann diese um 1 erhöht. Die zweite Version arbeitet genau umgekehrt. Zuerst wird der Wert erhöht, um ihn dann zurückzugeben.

```
<?
$x = $y = 20;
echo $x++;
echo "<br>";    //Zeilenumbruch!
echo ++$y;
?>
```

Listing 3.4: Inkrementierung

Ausgabe:

```
20
21
```

Dekrementierung

Das genaue Gegenteil der Inkrementierung ist die Dekrementierung. Es ist nicht schwer zu erraten, wie sich diese Art von Zuweisung aufbaut, wenn man das letzte Kapitel gelesen hat. Es geht darum, dass Variablen um den Wert 1 reduziert werden können. Anstatt

```
$x = $x - 1;
```

zu schreiben, bietet PHP wieder eine verkürzte Version an.

```
$x - -;
```

Auch hier gilt dasselbe wie für die Inkrementierung: Es gibt zwei Möglichkeiten eine Variable zu dekrementieren. Je nach Art wird die Reihenfolge von Zuweisung und Rückgabe bestimmt.

```
- - $x;
```

Die zweite Variante weist zuerst die Dekrementierung zu und gibt dann den Wert zurück.

```
<?
$x = $y = 20;

echo $x--;
echo "<br>";    //Zeilenumbruch!
echo --$y;
?>
```

Listing 3.5: Dekrementierung

Ausgabe:

```
20
19
```

Weitere Zuweisungsoperatoren

PHP bietet noch fünf weitere Zuweisungsoperatoren, die auf den fünf arithmetischen Operatoren basieren.

Beispiel	Beschreibung
$a + = 5;	Dem Wert von $a wird 5 hinzugefügt und das Ergebnis wird $a zugewiesen.

Tabelle 3.2: Weitere Zuweisungsoperatoren

Beispiel	Beschreibung
$a - = 5;	Dem Wert von $a wird 5 abgezogen und das Ergebnis wird $a zugewiesen.
$a * = 5;	Der Wert von $a wird mit 5 multipliziert und das Ergebnis wird $a zugewiesen.
$a / = 5;	Der Wert von $a wird durch 5 geteilt und das Ergebnis wird $a zugewiesen.
$a % = 5;	Dem Wert von $a wird durch 5 geteilt und der ganzzahlige Rest wird $a zugewiesen.

Tabelle 3.2: Weitere Zuweisungsoperatoren (Forts.)

Hier ein kleines Beispiel, das den Einsatz dieser Zuweisungsoperatoren demonstriert.

```
<?
$a = 20;
$a += 5;
echo $a;
echo "<br>";    //Zeilenumbruch
$a = 20;
$a /= 5;
echo $a;
?>
```

Listing 3.6: Weitere Zuweisungsoperatoren

Ausgabe:

```
25
4
```

3.3.3 Rangfolge von Operatoren

PHP kennt alle mathematischen Regeln und hält sich auch gewissenhaft daran. Die Rangfolge von Operatoren richtet sich demnach streng nach den mathematischen Vorschriften, wie Sie sie in der Schule gelernt haben.

- Punkt vor Strich (Modulo ist »Punkt«-Rechnung)
- Klammern werden entsprechend berücksichtigt
- Funktionen werden mit dem Wert verrechnet, den sie zurückgeben

- In- bzw. Dekrementierung steht vor der Punktrechnung
- Diese Grundregeln sollten Sie immer im Hinterkopf haben, damit Sie zu den gewünschten Ergebnissen gelangen.

```
<?
$y=1;
$x = 5 + 4 * (4 / ++$y);
echo $x;
?>
```

Listing 3.7: Rangfolge der Operatoren

Ausgabe:

```
13
```

3.3.4 Vergleichsoperatoren

Vergleichsoperatoren erlauben es, Ausdrücke miteinander zu vergleichen und so Informationen über sie zu erhalten. Dies ist gleichzeitig die Grundlage für logische Bedingungen, die es ermöglichen, Entscheidungen in einem Programm zu treffen und danach zu handeln.

Das Ergebnis eines mit Vergleichsoperatoren konstruierten Ausdrucks ist entweder TRUE oder FALSE.

Ausdruck	Beispiel	Ergebnis
gleich	`$x == $y`	TRUE, wenn die Werte gleich sind
größer als	`$x > $y`	TRUE, wenn $x größer als $y ist
kleiner als	`$x < $y`	TRUE, wenn $x kleiner als $y ist
kleiner gleich	`$x <= $y`	TRUE, wenn $x kleiner oder gleich $y ist
größer gleich	`$x >= $y`	TRUE, wenn $x größer oder gleich $y ist
ungleich	`$x != $y`	TRUE, wenn die Werte ungleich sind

Tabelle 3.3: Vergleichsoperatoren

Anstelle einzelner Variablen können natürlich auch komplexe Ausdrücke vor und hinter den Vergleichsoperatoren stehen. PHP berechnet dann zuerst die Ergebnisse, um sie miteinander zu vergleichen.

```
<?
$x = 5;
$y = 4;
echo $x > $y;
echo "<br>";    //Zeilenumbruch
echo 3 * $x -2 >= 4 * $y + 5;
echo "<br>";    //Zeilenumbruch
echo $x == $y +1;
?>
```

Listing 3.8: Vergleichsoperatoren

Ausgabe:

```
1
0
1
```

Es passiert oft, dass der Zuweisungsoperator = mit dem Vergleichsoperator == verwechselt wird. Diese Fehlerquelle sollten Sie unbedingt vermeiden, da es oft sehr knifflig ist diesen Fehler zu finden. Eine Zuweisung quittiert PHP immer mit einem TRUE, so dass spätere Bedingungen immer wahr bleiben. Gerade Anfänger treibt dieser Fehler schnell zur Verzweifelung.

3.3.5 Logische Operatoren

Logische Operatoren dienen zur Verknüpfung von booleschen Ausdrücken um den Wahrheitswert von kombinierten Aussagen zu erhalten. Der Wert von kombinierten Wahrheitswerten ist entweder TRUE oder FALSE.

Name	Beispiel	Ergebnis
und	`$ww1 and $ww2`	TRUE, wenn beide Werte wahr sind
oder	`$ww1 or $ww2`	TRUE, wenn einer der beiden Werte wahr ist
entweder oder	`$ww1 xor §ww2`	TRUE, wenn einer der beiden Werte wahr ist, aber nicht beide
nicht	`! $ww1`	TRUE, wenn der Wert falsch ist

Tabelle 3.4: Logische Operatoren

Die Variablen `$ww1` und `$ww2` stehen für zwei Wahrheitswerte, die entweder TRUE oder FALSE sein können. Die Operatoren können nach Belieben miteinander kombiniert und verschachtelt werden, um jede beliebige logische Aussage zu erschaffen.

Anstelle von `and` kann auch der Operator `&&` verwendet werden, der genau dieselbe Funktion hat. Anstelle von *or* bietet PHP den Operator `||` an, der ebenfalls genau dasselbe bewirkt. Der einzige Unterschied ist die Rangfolge dieser Operatoren.

- `||` und `&&` werden als Erstes berücksichtigt
- `and` kommt an zweiter Stelle
- `xor` an dritter
- und `or` an letzter Stelle
- Mit Klammern kann man diese Reihenfolge verändern

3.3.6 Übungen

Ü

1. Wozu brauchen wir Operatoren?
2. Schreiben Sie ein Programm, das das Quadrat einer Zahl berechnet.
3. Was ist der Unterschied zwischen der Inkrementierung und der Dekrementierung?
4. Erläutern Sie den Unterschied zwischen dem Operator = und dem Operator == .
5. Wird `or` oder `and` in der Rangfolge zuerst berücksichtigt?

3.4 Datentypen

Die Sprache PHP unterstützt sechs verschiedene Datentypen, die Sie aus anderen Programmiersprachen bereits kennen. Vier von dieses Datentypen sind Standarddatentypen. Arrays sind Feldtypen und ein Objekt ist eine vom Programmierer definierte Datenkonstruktion.

Datentyp	Beschreibung
`boolean`	TRUE oder FALSE
`integer`	Ganzzahl
`double`	Kommazahl
`string`	Zeichenkette
`array`	Feldtyp (Liste)
`object`	definiertes Objekt

Tabelle 3.5: PHP-Datentypen

Im Gegensatz zu vielen anderen Programmiersprachen geht PHP recht unkompliziert mit der Definition von Datentypen um. Die Zuweisung eines bestimmten Datentyps an eine Variable wird nicht verlangt, ganz im Gegenteil. PHP übernimmt diese Aufgabe automatisch und erkennt den verlangten Datentyp anhand des zugewiesenen Wertes.

Der PHP-Interpreter geht bei der Bestimmung des Datentyps nach folgenden Regeln vor:

- Jede ganze Zahl (ohne Komma!) wird als Ganzzahl interpretiert
- Jede Kommazahl ist ein `double`-Wert
- Zeichenketten müssen von Anführungszeichen umschlossen sein

PHP nutzt die amerikanische Punktnotation zur Darstellung und Berechnung von Kommazahlen. Sollten Sie also Kommazahlen in Ihren Programmen verwenden, dann setzen Sie anstelle des Kommas einen Punkt an der Stelle, wo der Nachkommabereich beginnt.

```
$var = 5,1;     //FALSCH!!!
$var = 5.1;     //richtig!
```

PHP bestraft den Einsatz der falschen Notation mit einer »Parse Error«-Fehlermeldung.

3.4.1 Rechenergebnisse

Werden zwei Variablen in einer Rechnung miteinander verknüpft, dann entsteht ein neuer Wert. Anhand der Datentypen der beteiligten Variablen wird der Datentyp des neuen Wertes bestimmt. Auch hier gibt es Regeln:

- Sind alle Werte `integer`-Werte, dann ist auch das Ergebnis eine Ganzzahl
- Sobald ein Wert als `double` gewertet wird, ist das Ergebnis auch eine Kommazahl (`double`)
- Strings können über den . – Operator mit anderen Werten verbunden werden. Das Ergebnis ist immer ein String

3.4.2 Typenumwandlung

PHP verwaltet alle Datentypen intern in einem Arbeitsspeicher. Sollte es nötig sein, wird eine Variable automatisch in einen anderen Datentyp umgewandelt, ohne dass der Programmierer etwas davon merkt. Dieses Vorgehen nennt man implizite Typenumwandlung.

```
<?
$var = 5;              //integer-Wert
$var = $var + 0.1;     //Umwandlung in double
echo $var;
?>
```

Listing 3.9: Implizite Typenumwandlung

Ausgabe:

```
5.1
```

Die Variable $var ist zuerst vom Typ `integer`, da ihr eine Ganzzahl zugewiesen wurde. Dann wird die Variable automatisch in eine Kommazahl (`double`) umgewandelt, da der Wert um 0.1 erhöht wurde.

Möchte man den Typ einer Variablen per Hand ändern, so ist dies über die explizite Typenumwandlung möglich. Dazu stellt man dem betreffenden Wert einfach den gewünschten Zieltyp in Klammern voran. Der Ausdruck wird automatisch umgewandelt.

```
$var = 20;   //integer
$var = (double)$var; //Umwandlung in double
```

PHP unterstützt folgende Umwandlungsoperatoren:

Operator	Beschreibung
`(int)`	Wandlung zu `integer`
`(integer)`	Wandlung zu `integer`
`(double)`	Wandlung zu `double`
`(float)`	Wandlung zu `double`
`(string)`	Wandlung zu `String`

Tabelle 3.6: Explizite Typumwandlung

Eine weitere Möglichkeit, einen Datentyp zu ändern, ist die Funktion `settype()`. Um einer Variable einen neuen Typ zuzuweisen, braucht die Funktion zwei Parameter. Einmal die Variable selbst und dann den neuen Zieltyp.

```
<?
$int = 20;
settype($int, "double");
echo $int;
?>
```

Listing 3.10: Typenumwandlung mit settype()

`settype()` unterstützt alle Datentypen in PHP. Sie werden einfach als String übergeben.

3.4.3 Datenverluste bei der Umwandlung

Es kann passieren, dass es je nach Art der Umwandlung zu Datenverlusten kommt. Das ist allerdings kein PHP-spezifisches Problem, sondern vielmehr eine logische Folge aus der Umwandlung. Das klassische Beispiel ist wahrscheinlich der Schritt von einer Kommazahl zu einem `integer`-Wert.

```
<?
$komma = 3.75;
settype($komma, "integer");
echo $komma;
?>
```

Listing 3.11: Datenverluste bei Typenumwandlung

Ausgabe:

```
3
```

Die Kommazahl `$komma` wird durch die Funktion `settype()` in eine `integer`-Variable umgewandelt. Da PHP keine Kommazahlen im `integer`-Format zulässt, wird der Nachkommateil einfach abgeschnitten.

Ein weiterer interessanter Punkt bei der Datenumwandlung ist die Typenänderung von einer Stringvariable zu einem Zahlentyp. Was auf den ersten Blick etwas unsinnig klingt, kann in PHP durchaus Sinn machen. Die Sprache unterstützt einige intelligente Mechanismen, die automatisch nach Zahlenwerten in einem String suchen. Diese Werte werden für den neuen Typ eingesetzt.

```
<?
$str = "24 Euro";
$str = (int)$str;
echo $str;
?>
```

Listing 3.12: Stringumwandlung

Ausgabe:

```
24
```

PHP setzt den Wert 24 automatisch für den neuen `integer`-Wert. Der Rest des Strings wird gelöscht.

3.4.4 Datentypen bestimmen

Im Laufe eines Programms ist es nicht immer einfach zu wissen, von welchem Datentyp eine Variable ist. PHP stellt eine Reihe von Funktionen zur Verfügung, die es erlauben, in jeder Situation den Typ einer Variablen zu bestimmen.

Mit der allgemein gehaltenen Funktion `gettype()`, die das genaue Gegenteil von `settype()` ist, kann man den Typ einer Variablen abfragen. Die Rückgabe der Funktion ist ein String, der den jeweiligen Datentyp enthält. Neben dieser Funktion gibt es für jeden Datentyp eine eigene Abfragefunktion, die entweder TRUE oder FALSE zurückgibt. Jede Funktion beginnt mit `is_`, gefolgt vom gewünschten Typ.

Funktion	Rückgabe
gettype()	String
is_double()	boolean
is_string()	boolean
is_array()	boolean
is_object()	boolean
is_bool()	boolean

Tabelle 3.7: Funktionen zur Bestimmung von Datentypen

```
<?
$var = "Hallo PHP!";
$int = 120;
$bool = TRUE;
echo "Der Typ von \$var ist " . gettype($var);
echo "<br>";
echo "Der Typ von \$int ist " . gettype($int);
echo "<br>";
echo "Der Typ von \$bool ist " . gettype($bool);
echo "<br>";
?>
```

Listing 3.13: Datentypen

Ausgabe:

```
Der Typ von $var ist string
Der Typ von $int ist integer
Der Typ von $bool ist boolean
```

Der Backslash vor den Variablen in der Funktion `echo` verhindert, dass PHP die Werte interpretiert und den Inhalt der Variablen ausgibt. Der Punkt vor `gettype()` verknüpft zwei Strings miteinander, um sie ausgeben zu können.

3.4.5 Übungen

Ü

1. Was ist an dieser Zuweisung falsch?

   ```
   $kommazahl = 4,5;
   ```

2. Was versteht man unter einer Typumwandlung?

3. Was gibt dieses Skript aus?

   ```
   <?
   $var = "Hallo PHP!";
   echo gettype($var);
   ?>
   ```

3.5 Variablen

In den letzten Kapiteln haben wir schon ausgiebig von Variablen Gebrauch gemacht, ohne sie eigentlich näher zu erläutern. Dieses Versäumnis werden wir jetzt nachholen.

Eine Variable symbolisiert im Prinzip nichts anderes als einen Teil des Speicherbereichs des Computers. Über die Variable haben wir die Möglichkeit Daten in diesem Bereich abzulegen und zu einem späteren Zeitpunkt wieder auszulesen. Die Größe des Speicherbereichs richtet sich nach dem Typ der Variable und wird von PHP automatisch bei einer Typänderung angepasst.

Jede Variable beginnt mit dem $-Zeichen und wird so von PHP erkannt. Daten ohne $-Zeichen sind Konstanten. Im Gegensatz zu anderen, stärker typorientierten Sprachen (C/C++, Java etc.) braucht in PHP keine Typzuweisung zu erfolgen. Der Interpreter weist den Typ automatisch anhand des Wertes der Variablen zu.

3.5.1 Namenskonventionen

Ein Variablenname muss mit einem Buchstaben oder einem Unterstrich beginnen. Das erste Zeichen darf keine Zahl sein, allerdings dürfen sonst Zahlen vorkommen. Variablen werden durch die Voranstellung eines $-Zeichens markiert.

Variablennamen können beliebig lang sein, dürfen aber keine Sonderzeichen enthalten. Auch deutsche Umlaute und das ß sind verboten. PHP unterscheidet zwischen der Groß- und Kleinschreibung, also ist die Variable $zahl eine ganz andere als $Zahl.

```
<?
$zahl = 10;
$Zahl = 20;
$nummer1 = "Dirk";
$_mein_Alter = 22.9;
echo $zahl;
echo "<br>";
echo $Zahl;
echo "<br>";
echo $nummer1;
echo "<br>";
echo $_mein_Alter;
?>
```

Listing 3.14: Variablen

Ausgabe:

```
10
20
Dirk
22.9
```

Die Typzuweisung geschieht automatisch und wird nach Bedarf auch automatisch angepasst.

3.5.2 Sichtbarkeit von Variablen

In PHP gilt generell die Regel, dass alle Variablen nur lokal sichtbar sind. Das bedeutet, dass Variablen, die in einer Funktion definiert worden sind, außerhalb dieser Funktion keine Gültigkeit besitzen. Hat PHP die Funktion komplett ausgeführt, dann wird die Variable aus dem Speicher gelöscht.

Dieses Vorgehen ist auch aus anderen Programmiersprachen bekannt und sollte nichts Neues sein. Auf der anderen Seite geht PHP allerdings noch einen Schritt weiter: Global definierte Variablen sind in Funktionen desselben Programms ebenfalls nicht sichtbar. Wenn man sich dieses Punktes nicht bewusst ist, kann es einen schnell zu Verzweiflung treiben. Ich spreche hier aus Erfahrung.

```
<?
$var = 120;
function func()
{
        $var = 200;
        echo $var;
}
func();
echo "<br>";
echo $var;
?>
```

Listing 3.15: Sichtbarkeit von Variablen

Ausgabe:

```
200
120
```

Auch wenn wir Funktionen noch nicht besprochen haben, wird das Prinzip dieses Listings schnell klar. Wir haben zwei Variablen, die denselben Namen tragen, aber aufgrund der Programmstruktur nichts miteinander zu tun haben. Das $var in der Funktion hat den Wert 200 und die globale Variable $var hat den Wert 120, wie die Ausgabe beweist. Die Variable wird nicht überschrieben.

Möchte man innerhalb einer Funktion auf eine globale Variable zugreifen, muss man diese global vereinbaren. Dies geschieht über das Schlüsselwort `global`.

```
<?
$var = 120;
function func()
{
        global $var;
        $var = 200;
        echo $var;
}
func();
echo "<br>";
echo $var;
?>
```

Listing 3.16: Das Schlüsselwort global

Ausgabe:

```
200
200
```

In diesem Beispiel wird durch das Einsetzen von `global` in der Funktion `func()` das Ergebnis grundlegend geändert. Die globale Variable `$var` wird überschrieben, da es sich jetzt in der Funktion um denselben Wert handelt.

Möchte man den Wert einer Variablen über die Lebensdauer einer Funktion hinaus erhalten, braucht man ein weiteres Schlüsselwort, nämlich `static`. Hiermit wird PHP gesagt, dass es die Variable im Speicher behalten soll, damit sie bei einem erneuten Aufruf der Funktion reaktiviert werden kann.

```
<?
function func()
{
        static $var1 = 0;
        $var2 = 0;
        $var1 = $var1 + 200;
        $var2 = $var2 + 200;
        echo "var1: $var1";
        echo "<br>";
        echo "var2: $var2";
}
func();
echo "<br>";
```

```
func();
echo "<br>";
func();
?>
```

Listing 3.17: Das Schlüsselwort static

Ausgabe:

```
var1: 200
var2: 200
var1: 400
var2: 200
var1: 600
var2: 200
```

In der Funktion werden zwei Variablen definiert, `$var1` und `$var2`. Die erste Variable ist statisch deklariert, das heißt, dass PHP sie im Speicher behält, auch wenn die Funktion ausgeführt wurde.

Der Unterschied zur »normalen« Variablenpolitik von PHP wird mit diesem Skript deutlich: `$var2` wird bei jedem neuen Aufruf von `func()` neu definiert und erhält damit immer den Wert 200 (da ihr Startwert 0 ist). Im Gegensatz dazu wird `$var1` aus dem Speicher gezogen und kann so mit ihrem alten Wert weiterarbeiten.

Jeder neue Funktionsaufruf erhöht den Wert von `$var1` um den Wert 200.

3.5.3 Funktionen zur Überprüfung von Variablen

Neben den bekannten Funktionen zur Überprüfung des Datentyps einer Variablen hat PHP auch einige andere Funktionen zur Kontrolle über Variablen geschaffen.

Die bekannteste Kontrollfunktion ist wohl `isset()`, mit der überprüft werden kann, ob eine Variable auch tatsächlich im Speicher existiert. Diese Kontrolle ist vor allem bei interaktiven Webseiten nötig, wenn man nicht weiß, welche Eingaben ein Surfer gemacht hat. Die Funktion gibt entweder TRUE oder FALSE zurück, je nachdem, wie das Ergebnis ausgefallen ist.

Die Anweisung `unset()` regelt die Angelegenheit, die bei `isset()` noch ungewiss ist. Mit `unset()` kann eine Variable aus dem Speicher entfernt werden. Die Funktion gibt TRUE zurück, wenn der Vorgang erfolgreich war.

Die Kontrolle mit der Funktion `empty()` kann dann interessant sein, wenn Sie nicht genau wissen, ob diese Variable im Programmverlauf gesetzt worden ist oder nicht. Die Funktion überprüft, ob die Variable »leer« ist, das heißt, ob sie auf dem Standardwert des Datentyps (0 oder »«) steht oder nicht. Die Rückgabe ist auch hier TRUE oder FALSE.

Die generelle Überprüfung eines Wertes kann mit der Funktion `is_numeric()` vorgenommen werden. Es wird überprüft, ob die Variable einen Zahlenwert (`integer`, `double`) gespeichert hat oder nicht. Je nach Ergebnis wird TRUE oder FALSE zurückgegeben.

```
<?
$var = 100;
echo is_numeric($var);
echo "<br>";
echo isset($var);
echo "<br>";
unset($var);
echo isset($var);
echo "<br>";
$var = "";
echo empty($var);
?>
```

Listing 3.18: Prüffunktionen

Ausgabe:

```
1
1
0
1
```

Das kleine Listing demonstriert den Einsatz aller besprochenen Prüffunktionen. Alle Überprüfungen wurden mit TRUE abgeschlossen, außer der dritten. Da wir die Variable zuvor mit `unset()` aus dem Speicher entfernt haben, kann PHP sie nicht mehr finden.

3.5.4 Weitere Vergleichsoperatoren

Seit der Version 4 kennt PHP zwei weitere Vergleichsoperatoren, mit deren Hilfe der Vergleich von Variablen noch weitergeführt werden kann. Bisher hatten wir nur den Wert zweier Variablen verglichen ohne auf den Typ Rücksicht zu nehmen. Das ändert sich, wenn wir die folgenden Operatoren verwenden.

Beispiel	Erklärung
$a === $b	TRUE, wenn sowohl der Wert als auch der Typ identisch ist
$a !== $b	TRUE, wenn der Wert oder der Typ unterschiedlich ist

Tabelle 3.8: Weitere Vergleichsoperatoren

Die Auswirkung dieser scheinbaren Haarspalterei zeigt sich im folgenden Listing.

```
<?
$var1 = 100;            //integer
$var2 = 100.0;  //double
echo $var1 == $var2;
echo "<br>";
echo $var1 === $var2;
?>
```

Listing 3.19: Weitere Vergleichsoperatoren

Ausgabe:

```
1
0
```

Der erste Vergleich wird von PHP als TRUE eingestuft, da beide Variablen denselben Wert haben. Der zweite Vergleich ist allerdings FALSE, da `$var2` im Gegensatz zu `$var1` von Typ `double` ist.

3.5.5 Konstanten

Konstanten sind im Gegensatz zu Variablen zur Laufzeit eines Programms nicht mehr veränderbar. Um eine Konstante zu vereinbaren, brauchen wir die Funktion `define()`. Wir übergeben der Funktion den Namen der zukünftigen Konstanten und den gewünschten Wert. Danach steht uns diese Konstante zur Verfügung.

```
<?
define ("pi", 3.1415326);
echo pi;
echo "<br>";
echo gettype(pi);
?>
```

Listing 3.20: Konstanten mit define()

Ausgabe:

```
3.1415326
double
```

Konstanten können nach der Deklaration genau wie Variablen behandelt werden. Die Typzuweisung erfolgt ebenfalls automatisch, wie das Listing zeigt.

PHP-Konstanten

Bei jedem Programmstart definiert PHP eine Reihe von Konstanten, die dem Programmierer zur Verfügung stehen. Sie geben Auskunft über das Skript, den PHP-Interpreter und den Server, auf dem das Skript läuft.

Konstante	Beschreibung
_ _ FILE _ _	Pfad und Dateiname des aktuellen Skripts
_ _ LINE _ _	aktuelle Zeile, die gerade interpretiert wird
PHP_VERSION	Version des PHP-Interpreters
PHP_OS	Name des Betriebssystems
E_ERROR	Fehlermeldung, wird von PHP zur Fehlerausgabe genutzt
E_WARNING	Warnung, wird von PHP zur Hinweisausgabe genutzt

Tabelle 3.9: PHP-Konstanten

```
<?
echo __FILE__;
echo "<br>";
echo __LINE__;
echo "<br>";
echo PHP_VERSION;
echo "<br>";
echo PHP_OS;
?>
```

Listing 3.21: PHP-Konstanten

Ausgabe:

```
c:\Programme\Apache Group\Apache\htdocs/gotophp4/listing16.php
4
4.0.4pl1
WIN32
```

Die Ausgabe des Listings wird sich natürlich ändern, je nachdem, auf welchem Rechner es gestartet wird. Diese Ausgabe verrät uns, dass mein Rechner ein Windowsrechner ist, auf dem ein Apache Server mit der PHP-Version 4.0.4 läuft. Die Datei heißt sinnigerweise listing16.php.

3.5.6 Übungen

Ü

1. Was ist nicht korrekt?

```
$heiß = TRUE;
$kalt = FALSE;
```

2. Was gibt dieses Skript aus?

```
<?
$temperatur = 12;
echo $Temperatur;
?>
```

3. Erläutern Sie den Unterschied zwischen einer lokalen und einer globalen Variable!
4. Was ist der Unterschied zwischen dem Vergleichsoperator === und dem Vergleichsoperator == ?
5. Was unterscheidet Konstanten von Variablen?

3.6 Strings

Eines der mächtigsten Werkzeuge der Programmiersprache PHP ist der Umgang mit Zeichenketten. Im Gegensatz zu vielen anderen Sprachen, die oft sehr stiefmütterlich mit Strings umgehen, bietet PHP eine reiche Auswahl an verschiedensten Funktionen, die einem die Arbeit rund um die Zeichenketten abnehmen.

Dieses Angebot ist allerdings auch bitter nötig, denn eines der häufigsten Probleme, die einem beim Erstellen von dynamischen Webseiten begegnen, ist der sichere Umgang mit Strings.

Oft müssen Texte von einer Codierung in eine andere übertragen werden, um sie korrekt im Web darstellen zu können. Dies ist nur eins von vielen Beispielen, das in anderen Programmiersprachen für Probleme sorgen könnte, aber mit PHP souverän gelöst werden kann.

3.6.1 Ausgabe von Strings

Die wohl am häufigsten verwendeten Funktionen in PHP sind Ausgabefunktionen, die Strings an den Browser des Clientrechners senden. Bisher haben wir immer den Ausdruck `echo` verwendet, um etwas an den Browser weiterzuleiten, aber PHP hat noch einiges mehr zu bieten.

Zur Ausgabe von Zeichenketten stehen folgende Funktionen zur Verfügung:

Beispiel	Beschreibung
`echo "Hallo Welt!";`	gibt den String aus
`print ("Hallo Welt");`	gibt den String aus (Klammern nötig)

Tabelle 3.10: Rudimentäre Ausgabefunktionen

Die Funktion `print()` ist in Anlehnung an Perl entstanden, das dieselbe Funktion zur Ausgabe von Strings verwendet. Ein weiterer Unterschied ist, dass `print()` nur ein Argument akzeptiert, während `echo` mehrere Variablen übernehmen kann.

Eine weitere Ausgabefunktion ist `chr()`, die es erlaubt, einzelne Zeichen anhand der Nummer des ASCII-Systems auszugeben. Die Funktion erwartet einen `integer`-Wert zwischen 0 und 255 und berechnet anhand dessen das dazugehörigen ASCII-Zeichen.

Diese Funktion macht es relativ simpel, eine Übersicht aller ASCII-Zeichen zu schaffen.

```
<?
echo "<table border=1><tr>";
$i = 0;
while($i <= 255)
{
        echo "<td>";
        echo chr($i);
        echo "</td>";
        if (($i % 25) == 0) {echo "</tr><tr>";}
        $i++;
}
echo "</tr></table>";
?>
```

Listing 3.22: Übersicht ASCII-Zeichen

Die hier verwendete Schleife sorgt dafür, dass die Funktion `chr()` genau 256-mal ausgeführt wird, bevor die HTML-Tabelle wieder geschlossen wird. Die Variable $i wird pro Durchgang um 1 erhöht und generiert so dynamisch die `integer`-Zahl für jedes ASCII-Zeichen.

3.6.2 Formatierte Ausgabe von Strings

Im Gegensatz zu `echo` und `print()`, die ihre Argumente einfach ausgeben, bietet PHP zwei weitere Funktionen, mit denen es möglich ist, Strings zu formatieren. Neben dem auszugebenden String wird der Funktion ein Formatierungsstring übergeben, der aus verschiedenen Steuerzeichen zusammengestellt wird. Anhand dieses Strings wird die Zeichenkette formatiert.

Formatierungszeichen	Funktion
%	keine Formatierung
b	Wert wird als integer behandelt
c	Wert wird als ASCII-Zeichen interpretiert
d	Wert wird als Dezimalzahl angezeigt
f	Wert wird als double angezeigt
o	Wert wird als Oktalzahl behandelt
s	Wert wird als String behandelt
x	Wert wird als Hexadezimal gewertet
X	wie x, aber mit Großbuchstaben

Tabelle 3.11: Formatierungszeichen von printf()

```
<?
$str1 = "Hallo";
$str2 = "Welt!";
$str3 = "<br>";
printf("%s%s%s", $str1, $str3, $str2);
?>
```

Listing 3.23: Formatierung mit printf()

Ausgabe:

```
Hallo
Welt!
```

Neben der Funktion `printf()` gibt es eine weitere Funktion mit dem Namen `sprintf()`, die auf demselben Prinzip aufbaut. Der Unterschied liegt darin, dass `sprintf()` das Ergebnis nicht an den Browser ausgibt, sondern als einen String zurückgibt. Das gibt einem die Möglichkeit das Ergebnis in einer Variablen zu speichern.

```
<?
$str1 = "Hallo";
$str2 = "Welt!";
$str3 = "<br>";
$erg = sprintf("%s%s%s", $str1, $str3, $str2);
echo $erg;
?>
```

Listing 3.24: Formatierung mit sprintf()

Ausgabe:

```
Hallo
Welt!
```

3.6.3 Funktionen zur Bearbeitung von Strings

Der einfachste Weg zwei Strings miteinander zu verbinden, ist der Punktoperator. Er wird in PHP speziell eingesetzt um Strings zu verknüpfen. Die Handhabung ist denkbar einfach:

```
<?
$str1 = "Hallo";
$str2 = " ";
$str3 = "Welt!";
$satz = $str1 . $str2 . $str3;
echo $satz;
?>
```

Listing 3.25: Der Punktoperator

Ausgabe:

```
Hallo Welt!
```

Die drei Strings werden einfach zusammengefasst und zu einem neuen String geformt.

Neben diesem klassischen Operator kennt PHP eine ganze Reihe weiterer Funktionen zur Stringbearbeitung. Unter anderem werden damit Probleme wie Groß- und Kleinschreibung gelöst.

Funktion	Beschreibung
strrev()	Invertiert einen String
strtolower()	Wandelt alles in Kleinbuchstaben um
strtoupper()	Wandelt alles in Großbuchstaben um
ucfirst()	Wandelt das erste Zeichen in einen Großbuchstaben um
ucwords()	Wandelt das erste Zeichen jedes Wortes in einen Großbuchstaben um
ord()	Gibt den ASCII-Wert für ein Zeichen zurück

Tabelle 3.12: Stringfunktionen

```
<?
$str = "Hallo Welt!";
echo strrev($str);
echo "<br>";
echo strtolower($str);
echo "<br>";
echo strtoupper($str);
echo "<br>";
echo ucwords("hallo welt!");
?>
```

Listing 3.26: Stringfunktionen

Ausgabe:

```
!tleW ollaH
hallo welt!
HALLO WELT!
Hallo Welt!
```

3.6.4 Funktion zur Zeichenersetzung

Fast schon unentbehrlich sind Funktionen, die einem das Suchen und Ersetzen von Sonderzeichen oder Zeichenketten abnehmen. Sie werden gerade im Konventionen-Wirrwarr des Internet sehr oft benötigt und spielen deshalb eine besondere Rolle.

Funktion	Beschreibung
`AddSlashes()`	setzt Backslashes vor Sonderzeichen
`StripSlashes()`	entfernt Backslashes vor Sonderzeichen
`ltrim()`	entfernt Leerzeichen am Anfang des Strings
`str_replace()`	ersetzt Teile des Strings durch einen Ersatzstring
`trim()`	entfernt Leerzeichen am Anfang und am Ende des Strings

Tabelle 3.13: Suchen und Ersetzen

Die Funktionen `AddSlashes()` und `StripSlashes()` sind vielleicht etwas erklärungsbedürftig. PHP benutzt eine Reihe von Sonderzeichen, um bestimmte Punkte in der Syntax zu markieren.

Möchte man gerade diese Zeichen in einem String verwenden, ohne dass PHP diese als Sonderzeichen erkennt und interpretiert, dann müssen sie maskiert werden. Das geschieht ganz einfach durch Backslashes vor dem jeweiligen Zeichen. Die beiden Funktion machen genau dies und maskieren automatisch alle betroffenen Sonderzeichen.

```
<?
$zahl = 100;
$str = "Der Name der Variable ist zahl, und sie hat den Wert $zahl!";
echo $str;
echo "<br>";
$str2 = addSlashes($str);
echo $str2;
?>
```

Listing 3.27: Sonderzeichenmaskierung

Ausgabe:

```
Der Name der Variable ist 'zahl', und sie hat den Wert 100!
Der Name der Variable ist \'zahl\', und sie hat den Wert 100!
```

Eine weitere nützliche Funktion ist `str_replace()`. Sie erlaubt es ‚Zeichenketten in einem String zu suchen und durch andere zu ersetzen. In Verbindung mit einer HMTL-Formatierung ist es zum Beispiel möglich, Suchworte in einem Text hervorzuheben.

```
<?
$string = "Und eines Tages hatten sie es satt, für andere Geld rein-
zuholen und dunkle Limousinen zu fahren. Sie wollten keine Sonnen-
brillen mehr tragen und elegante Lederkoffer an geheimen Orten
übergeben, um die Welt vor neuen Kriegen zu retten. Vorbei waren die
Zeiten von durchfeierten Nächten und luxuriösen Jachten, auf denen
immer neue Schönheiten auf sie warteten um sich zu vergnügen. Sie
wollten endlich selbstständig sein und auf eigenen Beinen stehen. Es
war genau die richtige Zeit für eine Änderung. Und als die Chance
kam, ergriffen sie diese. Eine neue Zeit brach an...";
echo $string;
echo "<br><br>";
$newString = str_replace("ei", "<font color=red><b>ei</b></font>",
$string);
echo $newString;
?>
```

Listing 3.28: Stringersetzung mit str_replace

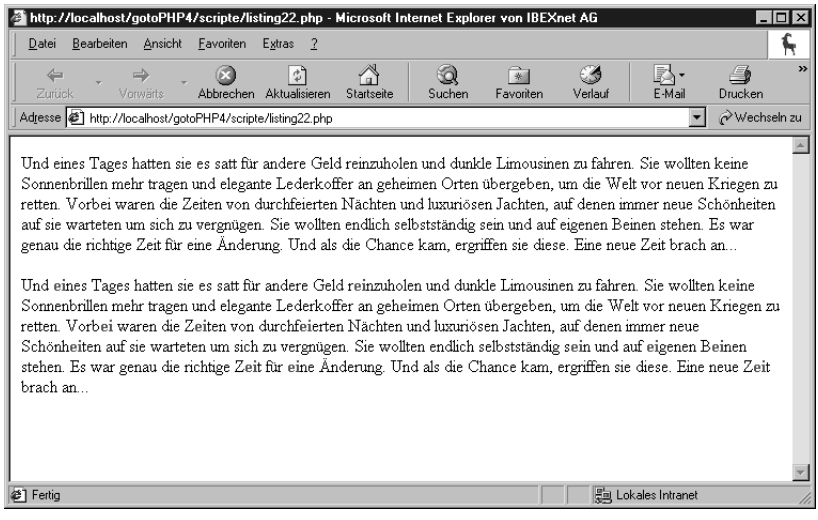


Abbildung 3.1: Ausgabe von Listing 22

Dieses Listing demonstriert das grundlegende Prinzip einer Suchmaschine, wie sie im Internet auf manchen Webseiten zu finden ist. Das Skript durchsucht den übergebenen Text (der in Realität entweder aus einer Datei oder einer Datenbank ausgelesen wird) auf die Buchstabenkombination »ei« und hebt diese dann farblich hervor.

Die Hervorhebung wurde ganz einfach durch den HTML-Tag <B> und <FONT> erreicht, die jeweils vor und hinter dem gesuchten String gesetzt werden. Es wurde also »ei« durch »<B><FONT COLOR=red>ei</B>« ersetzt.

Der Funktion `str_replace()` werden insgesamt drei Parameter übergeben, die für die Ersetzungen nötig sind. Der erste Parameter bestimmt den Suchstring in der Zeichenkette. Der zweite ist der Ersatzstring, der anstelle des Suchstrings eingesetzt wird. Der letzte Parameter ist der eigentliche Zielstring, der bearbeitet werden soll.

3.6.5 Suchfunktionen für Strings

Für einige Probleme der Webprogrammierung reicht es oft nicht aus, einen String als Ganzes zu betrachten, ohne seine Struktur genauer zu beleuchten. PHP bietet hier einige Funktionen, die Abhilfe schaffen und die es erlauben, Informationen über einen String zu erhalten.

Funktion	Beschreibung
`strpos()`	findet die Position einer Zeichenkette in einem String
`strrpos()`	findet das letzte Auftreten eines einzelnen Zeichens in einem String
`strstr()`	findet das erste Auftreten einer Zeichenkette und gibt diese mit allen folgenden Zeichen zurück
`substr()`	gibt einen Teilbereich eines Strings zurück

Tabelle 3.14: Informationen über Strings

Die Funktionen `strpos()` und `strstr()` sind vom Prinzip her ähnlich aufgebaut. Der Unterschied ist der Rückgabewert der Funktionen: `strpos()` gibt die exakte Position des Suchstrings als einen int-Wert zurück, während `strstr()` den String selbst ausgibt. Das folgende Beispiel erläutert dies:

```
<?
$string = "Und eines Tages hatten sie es satt, für andere Geld rein-
zuholen und dunkle Limousinen zu fahren.";
$pos1 = strpos($string, "sie", 10);
echo $pos1;
echo "<br>";
$pos2 = strstr($string, "sie");
echo $pos2;
echo "<br>";
```

```
$pos3 = strrpos($string, "G");
echo $pos3;
echo "<br>";
echo substr($string, $pos1, $pos3 - $pos1);
?>
```

Listing 3.29: Informationen über Strings

Ausgabe:

```
23
sie es satt, für andere Geld reinzuholen und dunkle Limousinen zu
fahren.
46
sie es satt, für andere
```

Der Funktion `strpos()` muss als dritter Parameter die Position angegeben werden, die im String gesucht werden soll. Möchten Sie den ganzen String durchsuchen, dann muss hier 0 eingegeben werden.

Die Funktion `substr()` braucht drei Parameter, um ausgeführt werden zu können. Der erste Wert ist der eigentliche String, der zerlegt werden soll. Der zweite Parameter ist die Startposition im String. Der dritte Wert gibt die Anzahl der Zeichen an, die `substr()` berücksichtigen soll, bevor der String abgeschnitten wird.

3.6.6 Strings vergleichen

Die folgenden Funktionen dienen zum Vergleich von Strings und geben ähnlich wie Vergleichsoperatoren entweder 0 für Gleichheit aus, -1 für den Fall, dass der erste String »kleiner« ist als der zweite oder eine 1 wenn der erste String »größer« ist als der zweite.

Funktion	Beschreibung
`strcmp()`	vergleicht zwei Strings
`strcasecmp()`	vergleicht zwei Strings unabhängig von der Groß- und Kleinschreibung
`strnatcmp()`	vergleicht zwei Strings unter Berücksichtigung von Zahlen
`strnatcasecmp()`	vergleicht zwei Strings unter Berücksichtigung von Zahlen, unabhängig von Groß- und Kleinschreibung

Tabelle 3.15: Vergleichsfunktionen für Strings

Der Vergleich von Strings geht in PHP einen Schritt weiter, als es im ersten Augenblick scheint. Es wird hier nicht nur zwischen »gleich« und »ungleich« unterschieden, sondern auch zwischen »größer« und »kleiner«.

Diese Wertigkeit ist uns aus dem Bereich der Zahlen bekannt und kann hier recht einfach übernommen werden. PHP wertet für diese Funktionen jeden einzelnen Buchstaben in einem String. Die Reihenfolge ist hierbei durch das Alphabet bestimmt, so dass ein a der kleinste und ein z der größte Buchstabe ist.

Groß- und Kleinschreibung wird bei zwei der Funktionen ebenfalls berücksichtigt. Großbuchstaben werden vor den Kleinbuchstaben gewertet, also hat `strcmp("A", "a")` den Wert –1.

```
<?
$str1 = "abc";
$str2 = "bcd";
echo strcmp($str1, $str2);
echo "<br>";
$str1 = "Olympia2002";
$str2 = "Olympia2000";
echo strnatcmp($str1, $str2);
?>
```

Listing 3.30: Stringvergleiche

Ausgabe:

```
-1
1
```

3.6.7 Übungen

1. Was ist ein String?
2. Was macht der Punktoperator?
3. Welchen Sinn macht eine mächtige Stringbibliothek in PHP?
4. Was macht die Funktion `str_replace()`?
5. Welche Kriterien werden bei Stringvergleichen angelegt?

3.7 Arrays

Arrays gehören zu den wichtigsten Datenstrukturen in PHP, man kann sie sich am einfachsten als eine Art Liste vorstellen. Ein Array ist kein einzel-

ner Wert, sondern vielmehr eine Zusammenfassung vieler Werte, die in einer gemeinsamen Struktur gespeichert werden.

Im Gegensatz zu stärker typorientierten Sprachen wie C/C++ oder Java ist die Handhabung von Arrays in PHP recht einfach, da es möglich ist, Werte von unterschiedlichen Datentypen im selben Array zu speichern.

Ein weiterer Punkt ist die Größe eines Arrays, die in PHP dynamisch vom Interpreter angepasst wird. Liest man über ein Array hinaus, führt das nicht zum Exitus des Programms, sondern man erhält lediglich eine Warnung.

3.7.1 Nummerische Arrays

Die einfachste Form eines Arrays ist ein nummerisches Array, das alle Elemente über einen durchnummerierten Index anspricht. Genau wie eine Variable wird ein Array durch ein $-Zeichen am Anfang der Bezeichnung gekennzeichnet. Die Namenskonventionen richten sich nach denselben Regeln.

Der Unterschied liegt im schon angesprochenen Index, der die einzelnen Elemente identifiziert. Um ein Element gezielt anzusprechen, reicht es, den jeweiligen Index in eckigen Klammern am Ende des Arrays anzugeben. PHP zählt von 0 an automatisch aufwärts.

Die Initialisierung eines nummerischen Arrays geschieht automatisch bei der Wertzuweisung. Ein einfaches Beispiel ist die altbekannte Einkaufsliste, die eine Reihe von Einträgen speichert.

```
<?
$liste[0] = "Milch";
$liste[1] = "Zucker";
$liste[2] = "Kaffee";
$liste[3] = "Brot";
$liste[4] = "Butter";
echo $liste[1];
echo "<br>";
echo $liste[3];
echo "<br>";
echo $liste[4];
echo "<br>";
echo $liste[0];
echo "<br>";
echo $liste[2];
?>
```

Listing 3.31: Nummerisches Array

Ausgabe:

```
Zucker
Brot
Butter
Milch
Kaffee
```

Über den jeweiligen Index können die einzelnen Elemente ausgelesen und wieder gesetzt werden. Ein Arrayelement kann genau wie eine Variable behandelt werden, das heißt es kann zum Beispiel als Parameter in einer Funktion dienen oder mit Operatoren verknüpft werden.

```
<?
$liste[0] = "Milch";
$liste[1] = "Zucker";
$liste[2] = "Kaffee";
$liste[3] = 17;
$liste[4] = 4;
echo "Das 3. Element des Arrays ist vom Typ: " . gettype($liste[2]);
echo "<br>";
echo "Ich trinke meinen ". $liste[2] ." mit ". $liste[0] .", aber
ohne ". $liste[1] ."!";
echo "<br>";
echo $liste[3] / $liste[4];
?>
```

Listing 3.32: Arbeiten mit Arrays

Ausgabe:

```
Das 3. Element des Arrays ist vom Typ: string
Ich trinke meinen Kaffee mit Milch, aber ohne Zucker!
4.25
```

Ein Array kann beliebig viele Elemente haben, PHP setzt hier keine Grenzen. Auch muss die maximale Anzahl der Elemente nicht vorher festgelegt werden, da der PHP-Interpreter die Größe dynamisch anpassen kann.

Das Schlüsselwort array

PHP kennt das Schlüsselwort `array`, mit dem es ebenfalls möglich ist, ein Array zu initialisieren. In diesem Fall werden die Elemente des neuen Arrays in Klammern hinter dem Schlüsselwort angegeben.

```
$liste = array("Milch", "Zucker", "Kaffee", "Brot");
```

Es ist egal, auf welche Weise man ein Array initialisiert, denn das Ergebnis ist genau dasselbe. Der Vorteil des Schlüsselworts `array` liegt in der Möglichkeit ein Leer-Array zu schaffen, das auf anderem Wege nicht ohne weiteres möglich ist.

```
$liste = array();
```

Solche Arrays werden zum Beispiel gebraucht, um sie im Laufe des Programms dynamisch füllen zu können.

3.7.2 Assoziative Arrays

Assoziative Arrays werden nicht über einen durchnummerierten Index verwaltet, sondern über bestimmte Schlüsselwörter, die einzelne Elemente kennzeichnen. Dieser Key sollte den Inhalt des Elements beschreiben, so dass eine assoziative Verbindung entsteht.

Die Syntax ist wie bei nummerischen Arrays, mit dem Unterschied, dass anstelle der fortlaufenden Indexnummer Strings verwendet werden.

```
<?
$liste["vorname"] = "Dirk ";
$liste["nachname"] = "Ammelburger";
$liste["email"] = "dirk@ammelburger.de";
$liste["web"] = "http://www.kulturbrand.de";
$liste["alias"] = "dok";
echo $liste["vorname"];
echo $liste["nachname"];
echo "<br>";
echo $liste["email"];
?>
```

Listing 3.33: assoziative Arrays

Ausgabe:

```
Dirk Ammelburger
dirk@ammelburger.de
```

Um auf ein bestimmtes Element in einem assoziativen Array zuzugreifen, benötigt man den jeweiligen Schlüsselstring. Der Vorgang ist derselbe wie bei nummerischen Arrays.

Assoziative Arrays können ebenfalls über das Schlüsselwort `array` angelegt werden. Die jeweiligen Schlüsselstrings und die dazugehörigen Elemente werden über den Operator => miteinander verbunden.

```
$liste = array("vorname" => "Dirk", "nachname" => "Ammelburger",
"email" => "dirk@ammelburger.de");
```

Der besseren Übersicht wegen ist es allerdings empfehlenswert, die Schlüssel-Wert-Paare untereinander zu schreiben.

```
$liste = array(
"vorname" => "Dirk",
"nachname" => "Ammelburger",
"email" => "dirk@ammelburger.de",
"web" => "http://www.kulturbrand.de"
);
```

Der Einsatz von assoziativen Arrays ist immer dann sehr nützlich, wenn man mit großen Datenmengen arbeitet, die dazu neigen, unübersichtlich zu werden. Es ist immer leichter, sich sinnvolle Schlüsselwörter zu merken, als abstrakte Zahlenkombinationen.

3.7.3 Arrayfunktionen

PHP stellt eine Reihe von verschiedenen Funktionen zur Verfügung, die einem die Arbeit mit Arrays sehr vereinfachen. Die meisten Funktionen behandeln das Sortieren und das Navigieren innerhalb eines Arrays. Einige andere Funktionen erlauben es, Informationen über Arrays zu erhalten.

Funktion	Beispiel	Beschreibung
`count()`	`$erg = count($liste)`	gibt die Anzahl der Elemente wieder
`array_count_values()`	`$arr = array_count_values($liste);`	zählt die Häufigkeit des Auftretens aller Elemente und gibt das Ergebnis in einem Array wieder
`array_keys()`	`$arr = array_keys($liste)`	gibt alle Schlüsselwörter eines assoziativen Arrays in einem weiteren Array wieder

Tabelle 3.16: Arrayfunktionen

Funktion	Beispiel	Beschreibung
array_values()	$arr = array_values($liste)	gibt alle Elemente eines assoziativen Arrays in einem weiteren Array wieder
array_reverse()	$arr = array_reverse($liste)	dreht ein Array herum
in_array()	$bool = in_array($str, $liste)	Gibt TRUE zurück, wenn der Wert im Array vorkommt.

Tabelle 3.16: Arrayfunktionen

```
<?
$liste = array(
"Brot", "Butter", "Käse", "Kaffee", "Zucker", "Brot",
"Kaffee", "Milch", "Zucker", "Brot", "Wurst", "Käse");
echo count($liste);
echo "<br>";
$arr = array_count_values($liste);
echo "Das Element ".$liste[0]." kommt " . $arr[$liste[0]]," mal
vor!";
echo "<br>";
echo "Das Element ".$liste[3]." kommt " . $arr[$liste[3]]," mal
vor!";
?>
```

Listing 3.34: Arrayfunktionen im Einsatz

Ausgabe:

```
12
Das Element Brot kommt 3 mal vor!
Das Element Kaffee kommt 2 mal vor!
```

Das Ergebnis der Funktion `array_count_values()` ist ein assoziatives Array, das als Schlüssel die Elemente des geprüften Arrays besitzt. Über diese Schlüssel kann dann die Anzahl jedes einzelnen Elements abgerufen werden.

Der Arrayzeiger

Neben der Möglichkeit, einzelne Elemente in einem Array direkt über den Index oder das Schlüsselwort anzusprechen, gibt es noch den Weg über den so genannten Arrayzeiger. Er markiert die Stelle, an der PHP im Array gerade steht und bereit ist die Daten zu manipulieren.

Standardmäßig steht dieser Zeiger auf der ersten Position eines Arrays, also 0. Er wird durch Lese- und Schreibaktionen automatisch verschoben und markiert so immer den letzten Stand der Dinge. Über bestimmte Funktionen ist es möglich, den Zeiger direkt anzusprechen und zu beeinflussen.

Funktion	Beschreibung
`current()`	gibt das Element aus einem Array zurück, auf das der Arrayzeiger zeigt
`each()`	gibt das nächste Schlüssel-Wert-Paar eines assoziativen Arrays zurück
`reset()`	setzt den Arrayzeiger auf das erste Element des Arrays
`end()`	setzt den Arrayzeiger auf das letzte Element des Arrays
`key()`	gibt den aktuellen Schlüssel des Elements wieder
`next()`	setzt den Zeiger um ein Element weiter vor
`prev()`	setzt den Zeiger um ein Element weiter zurück

Tabelle 3.17: Navigationsfunktionen im Array

```
<?
$liste = array(
"Brot", "Butter", "Käse", "Kaffee", "Zucker", "Brot",
"Kaffee", "Milch", "Zucker", "Brot", "Wurst");
echo current($liste);
next($liste);
echo "<br>";
echo current($liste);
echo "<br>";
end($liste);
echo current($liste);
echo "<br>";
reset($liste);
echo current($liste);
?>
```

Listing 3.35: Funktionen für die Navigation in einem Array

Ausgabe:

```
Brot
Butter
Wurst
Brot
```

Die Navigationsfunktionen sind alle relativ einfach gestrickt, so dass der Umgang mit ihnen intuitiv möglich ist. Es ist allerdings immer wichtig, im Hinterkopf zu behalten, dass das erste Element eines Arrays auf der Position 0 sitzt. Ein häufiger Fehler ist die Annahme, dass

```
reset($liste);
echo current($liste)
```

gleichzusetzen wäre mit

```
echo $liste[1];
```

Das ist natürlich verkehrt! Allerdings sitzt der Glaube, dass alles mit 1 beginnen muss, sehr tief, so dass uns das Unterbewusstsein oft einen Streich spielt.

Sortierfunktionen

Eine recht häufige Notwendigkeit bei der Arbeit mit Arrays ist das Sortieren der Elemente. PHP bietet einige Funktionen, die es erlauben, nach verschiedenen Kriterien sortieren zu lassen. Je nach Art des Arrays kann man nach den Elementen oder nach den Schlüsseln der Elemente sortieren.

Funktion	Beschreibung
`sort()`	sortiert ein Array alphabetisch anhand der Elemente
`rsort()`	sortiert ein Array gegen das Alphabet anhand der Elemente
`ksort()`	sortiert ein assoziatives Array nach dem Alphabet anhand der Schlüssel
`krsort()`	sortiert ein assoziatives Array gegen das Alphabet anhand der Schlüssel

Tabelle 3.18: Sortierfunktionen für Arrays

```
<?
$liste = array(
"Käse", "Kaffee", "Zucker", "Jogurt",
"Sahne", "Milch", "Salz", "Brot", "Butter");
sort($liste);
for($i = 0; $i < count($liste); $i++)
{
echo $liste[$i];
echo " ";
}
```

```
echo "<br>";
sort($liste);
for($i = 0; $i < count($liste); $i++)
{
echo $liste[$i];
echo " ";
}
?>
```

Listing 3.36: Sortierfunktion im Einsatz

Ausgabe:

```
Brot Butter Jogurt Kaffee Käse Milch Sahne Salz Zucker
Zucker Salz Sahne Milch Käse Kaffee Jogurt Butter Brot
```

Um die Ausgabe des kompletten Arrays zu vereinfachen, habe ich in diesem Listing ein wenig vorgegriffen und zwei Schleifen zum Einsatz gebracht. Ihre Aufgabe ist es lediglich, den kompletten Inhalt der Arrays auszugeben und so den Effekt der Sortierfunktionen zu demonstrieren. Schleifen und andere Programmstrukturen werden in Kapitel 4 des Buches behandelt.

Die Funktionen für assoziative Arrays funktionieren nach demselben Prinzip wie die gerade vorgestellten Sortierfunktionen. Ihre Kriterien beziehen sich allerdings auf die Schlüssel der Arrays und nicht auf die Elemente.

Weitere nützliche Arrayfunktionen

Der nun folgende Abschnitt beschreibt einige Arrayfunktionen, die nicht in die letzten Kapitel gepasst haben. Sie sind trotzdem zu wichtig, um sie einfach außer acht zu lassen.

Funktion	Beispiel	Beschreibung
`implode()`	`$str = implode(" ", $liste);`	verbindet alle Arrayelemente zu einem String, in dem sie durch Trennzeichen unterteilt sind
`explode()`	`$arr = explode(" ", $str);`	zerlegt einen String an bestimmten Trennzeichen und gibt das Ergebnis als Array zurück
`range()`	`$arr = range(1, 100);`	erstellt ein Array als integer-Zahlen. Anfangs- und Endwert werden als Parameter übergeben

Tabelle 3.19: weitere Arrayfunktionen

Funktion	Beispiel	Beschreibung
shuffle()	shuffle($liste);	mischt die Elemente zufällig durch
array_merge()	$arr = array_merge($liste1, $liste2);	verbindet zwei Arrays zu einem Neuen
array_pop()	$var = array_pop($liste);	entfernt das letzte Element eines Arrays und gibt es zurück
array_push()	array_push($liste, $elem);	fügt einem Array Elemente hinten an
array_shift()	$var = array_shift($liste);	entfernt das erste Element eines Arrays und gibt es zurück
array_unshift()	array_unshift($liste , $elem);	fügt einem Array Elemente vorne an

Tabelle 3.19: weitere Arrayfunktionen (Forts.)

Die beiden interessantesten Funktionen sind wohl `implode()` und `explode()`, die sich in vielen Programmen als unentbehrlich erwiesen haben. Ihre Einsatzmöglichkeiten sind sehr flexibel, darum möchte ich speziell hierzu ein kleines Beispiel geben.

```
<?
$liste = array(
"Käse", "Kaffee", "Zucker", "Jogurt",
"Sahne", "Milch", "Salz", "Brot", "Butter");
$str = implode(" ", $liste);
echo $str;
echo "<br>";
$str2 = "Kaffee und Kuchen und Bier und Wasser und Brot und Jogurt
und vieles mehr!";
$liste2 = explode(" und ", $str2);
for($i = 0; $i < count($liste2); $i++)
{
echo $liste2[$i];
echo "<br>";
}
?>
```

Listing 3.37: implode() und explode()

Ausgabe:

```
Käse Kaffee Zucker Jogurt Sahne Milch Salz Brot Butter
Kaffee
Kuchen
```

```
Bier
Wasser
Brot
Jogurt
vieles mehr!
```

Die erste `implode()`-Anweisung setzt das Array `$liste` zu einem String zusammen und verbindet die Elemente mit Leerzeichen. Der dadurch entstandene String kann gut lesbar ausgegeben werden.

In einem zweiten Schritt wird ein weiterer String mit `explode()` zerlegt. Als Trennzeichen haben wir hier das Wort »und« gewählt, das von zwei Leerzeichen flankiert wird. So ist es möglich, ein Array zu erzeugen, dass alle Elementen des Satzes enthält, ohne die störenden Konjunktionen zu berücksichtigen.

Die angegebenen Trennzeichen werden von `explode()` gelöscht und nicht mehr verwendet. Sie dienen lediglich als Festlegung der »Schnittstellen«.

Der Zufallsalgorithmus der Funktion `shuffle()` beruht genau wie bei anderen Zufallsfunktionen auf der aktuellen Systemzeit des Rechners. In Verbindung mit der Funktion range() kann sie als Grundlage für Spielesoftware im Internet dienen.

```
<?
$liste = range(1, 121);
//Ausgabe über implode!
echo implode(" ", $liste);
echo "<br>";
shuffle($liste);
//Ausgabe über implode!
echo implode(" ", $liste);
?>
```

Listing 3.38: range() und shuffle()

Ausgabe:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
114 115 116 117 118 119 120 121
```

```
103 48 118 114 99 1 107 23 104 20 82 72 16 95 111 116 62 106 27 19 31
2 94 52 3 28 47 17 12 58 83 90 22 46 24 4 80 5 117 66 115 43 79 75 84
113 56 54 119 93 6 40 64 18 81 21 69 32 102 100 25 65 30 45 26 74 61
53 86 120 29 8 70 7 121 57 76 97 88 15 108 14 9 63 59 13 60 39 10 89
67 11 55 41 34 33 42 77 51 73 71 50 44 98 105 36 38 112 110 101 109
68 35 78 37 92 49 85 96 91 87
```

Listing 38 könnte die Grundlage für ein Puzzlespiel mit 11 x 11 (= 121) Teilen sein, die vor Spielbeginn gemischt werden müssen. Diese Aufgabe übernimmt die Funktion `shuffle()`, die die Reihenfolge im Array ordentlich durcheinander bringt. Die Ausgabe des Listings ist natürlich bei jedem Aufruf anders, ganz wie der Zufall es will.

3.7.4 Globale Variablen über das Array $GLOBALS

Globale Variablen haben wir bereits in einem der letzten Kapitel besprochen. Wir hatten festgestellt, dass der Zugriff auf globale Variablen aus einer Funktion heraus nur eingeschränkt möglich ist.

Neben dem Schlüsselwort `global`, das eine Funktionsvariable global definiert, bietet PHP den Zugriff auf globale Variablen auch über ein Systemarray an. Das Array `$GLOBALS` speichert alle globalen Variablen assoziativ, so dass sie über den Namensschlüssel aufgerufen werden können.

```
<?
//Funktion
function test()
{
        echo $GLOBALS["var"];
        echo "<br>";
        echo $GLOBALS["str"];
}
//globale Variablen
$var = 20;
$str = "Hallo PHP!";
//Funktionsaufruf
test();
?>
```

Listing 3.39: Das Array $GLOBALS

Ausgabe:

```
20
Hallo PHP!
```

3.7.5 Mehrdimensionale Arrays

Alle bisher besprochenen Arrays waren eindimensional, das heißt, sie bestanden aus einer einzelnen Reihe von Daten, die aller untereinander bzw. nebeneinander gespeichert waren.

Kaffee
Milch
Zucker
Brot
Käse
...

Tabelle 3.20: Eindimensionales Array

Für einfache Listen ist die Form sehr vorteilhaft. Möchte man die gespeicherten Daten allerdings in eine gewisse Struktur bringen, dann wird die Sache schnell unübersichtlich.

Ein häufiges Beispiel ist die Konstruktion eines Kontaktdaten-Arrays, das Namen und Adressen vieler Personen speichern soll. Würde man diese Daten alle untereinander speichern, würde es schnell kompliziert werden, den jeweiligen Index eines Datensatzes zu ermitteln. Die Lösung ist ein zweidimensionales Array, das sich wie eine Tabelle darstellt.

Meier	Hans	Kleinstr. 7	Neustadt	0123 / 456789
Müller	Tobias	Großstr. 11	Altstadt	0987 / 543221
Schmidt	Ernst	Rosenweg 3	Großstadt	02345 / 444333
Huber	Gustav	Holzweg 134	Kleinstadt	0666 / 3342245
...	...	...	...	...

Tabelle 3.21: Zweidimensionales Array

Jeder zusammenhängende Datensatz steht jetzt schön sortiert in einer Zeile. Die Navigation in einem solchen Array ist jetzt allerdings ein wenig komplizierter, da wir uns über zwei Ebenen bewegen müssen.

Die Lösung liegt in der Betrachtung der Sache. Im Prinzip haben wir hier nichts anderes als ein Array, das aus einer Reihe weiterer Arrays besteht, die die Daten enthalten. PHP unterstützt diese Konstruktion, indem einfach ein zweiter Index (für das zweite Array) angehängt wird. Man übergibt dem Schlüsselwort `array` einfach weitere Arrays anstatt einfacher Variablen.

```
<?
$liste1 = array("Meier","Hans","Kleinstr. 7","Neustadt","0123 /
456789");
$liste2 = array("Müller","Tobias","Großstr. 11","Altstadt","0987 /
543221");
$liste3 = array("Schmidt","Ernst","Rosenweg 3","Großstadt","02345 /
444333");
$liste4 = array("Huber","Gustav","Holzweg 134","Kleinstadt","0666 /
3342245");
$array2D = array($liste1, $liste2, $liste3, $liste4);
echo $array2D[2][3];
echo "<br>";
echo $array2D[1][2];
echo "<br>";
echo $array2D[0][3];
echo "<br>";
echo $array2D[2][0];
?>
```

Listing 3.40: Zweidimensionales Array

Ausgabe:

```
Großstadt
Großstr. 11
Neustadt
Schmidt
```

Die einzelnen Arrays können auch direkt zugewiesen werden, wie das folgende Listing beweist.

```
<?
$array2D = array(
array("Meier","Hans","Kleinstr. 7","Neustadt"),
array("Müller","Tobias","Großstr. 11","Altstadt"),
array("Schmidt","Ernst","Rosenweg 3","Großstadt"),
array("Huber","Gustav","Holzweg 134","Kleinstadt")
);
?>
```

Listing 3.41: Direkte Zuweisung von Arrays

In Zusammenarbeit mit Datenbanken erhält man oft mehrdimensionale Arrays als Ergebnis einer Anfrage zurück. Dies hängt mit der Struktur der Datenverwaltung zusammen, die SQL-Anfragen mit einer Reihe von Datentupeln beantworten. Wie diese Problematik gelöst wird, sehen Sie im zweiten Teil des Buches.

Ein Array kann im Prinzip beliebig viele Dimensionen haben, auch wenn es schwierig ist, einen sinnvollen Einsatz für ein Array mit mehr als drei Dimensionen zu finden. Technisch gesehen muss für jede weitere Dimension einfach ein weiterer Index angehängt werden. Die Verschachtelung verläuft genauso wie für zweidimensionale Arrays.

3.7.6 Übungen

Ü

1. Was ist ein Array?
2. Wie werden Arrays erzeugt? Wie kann man die Elemente ansprechen?
3. Erläutern Sie den Unterschied zwischen einem nummerischen und einem assoziativen Array.
4. Was gibt das folgende Listing aus?

```
<?
$liste = array(
"Brot", "Butter", "Käse", "Kaffee", "Zucker", "Brot","Kaffee",
"Milch", "Zucker", "Brot");
end($liste);
echo current($liste);
?>
```

5. Welche Beziehung verknüpft die Funktionen implode() und explode()?
6. Welche besondere Funktion hat das Array $GLOBALS?
7. Was sind mehrdimensionale Arrays und wo finden sie Verwendung?

4 Programmstrukturen

Jedes Programm besteht aus einer Reihe von Anweisungen, die durch ein Semikolon abgeschlossen werden. Eine Anweisung kann eine einfache Zuweisung sein, aber auch ein komplexer Funktionsaufruf, der viele Werte im Programm beeinflusst. Diese Grundlagen wurde bereits in einem der vorangehenden Kapitel geklärt.

Die Programme, die wir bisher geschrieben haben, waren sehr klein und brauchten in den meisten Fällen keine besondere Struktur um zu funktionieren. Das wird sich jetzt ändern. PHP bietet wie alle Programmiersprachen eine große Anzahl von Möglichkeiten, Programme zu gliedern, sinnvoll aufzuteilen, zusammenzufassen und effektiv zu gestalten. Dies geschieht über so genannte Sprachkonstrukte.

Diese Sprachkonstrukte erlauben es zum Beispiel, die Reihenfolge der Abarbeitung von Anweisungen festzulegen oder von Bedingungen abhängig zu machen. Schleifenkonstruktionen ermöglichen es, Programmteile dynamisch zu generieren und den Quellcode klar zu gliedern.

Die im Folgenden vorgestellten Sprachkonstrukte orientieren sich in großen Teilen an den Sprachen C/C++ und Java. In einige Bereichen ergeben sich allerdings Besonderheiten, die von der logischen Struktur der Skriptsprache PHP abhängig sind. Im Gegensatz zu Java und C/C++ ist PHP eine reine Sprache, die nicht für einfache Anwendungen auf lokalen Systemen geschaffen wurde.

4.1 Kapitelübersicht

- Bedingungen – TRUE und FALSE im Einsatz
- Schleifen – Kontrollierte Wiederholungen
- Funktionen – Strukturierte Unterprogramme

4.2 Bedingungen

Bedingungen gehören zu den wichtigsten Steuerstrukturen in einer Programmiersprache. Sie ermöglichen es, im Programmverlauf Entscheidungen zu treffen und so den weiteren Verlauf zu beeinflussen.

Die Grundlage für Entscheidungen haben wir bereits mit verschiedenen Operatoren besprochen, die boolesche Werte zurückgeben. Eine Bedingung überprüft, ob ein Ausdruck TRUE (1) oder FALSE (0) ist und ermöglicht so, eine klare Entscheidung zu treffen.

4.2.1 Blöcke

Die Grundvoraussetzung für die logische Strukturierung eines Programms ist die sinnvolle Aufteilung von Programmteilen. Dies geschieht mithilfe von so genannten Blöcken, die Anweisungen zusammenfassen. Ein Block wird immer von geschweiften Klammern umgeben {}.

```
{
$var = 3;
echo "Dies ist ein Block! Der Wert von \$var ist ".
var;
}
```

Für sich genommen ist ein Block wenig sinnvoll, da er nicht mehr als eine optische Aufteilung ermöglicht. Für eine konkrete Steuerung des Programms brauchen wir weitere Anweisungen, die wir jetzt besprechen werden.

4.2.2 if-Bedingungen

Die `if`-Bedingung erlaubt es, einfache logische Ausdrücke zu überprüfen und das Ergebnis auszuwerten. Die Syntax entspricht der von C/C++ und ist einfach zu handhaben.

```
if (Bedingung) {Anweisungen}
```

Dem Schlüsselwort `if` folgt ein Ausdruck, der entweder TRUE oder FALSE zurückgeben muss. Ist er TRUE, dann wird der nachfolgende Anweisungsblock ausgeführt, wenn nicht, dann wird er übersprungen. Da Bedingungen ein Sprachkonstrukt sind, werden sie nicht von einem Semikolon abgeschlossen. Das gilt im Übrigen für alle folgenden Konstrukte.

```
<?
$erg = 1000 / 5;
if($erg == 200) {echo "1000 geteilt durch 5 ist 200!";}
if($erg == 250) {echo "1000 geteilt durch 5 ist 250!";}
?>
```

Listing 4.1: Die if-Bedingung

Ausgabe:

```
1000 geteilt durch 5 ist 200!
```

Zu diesem Zeitpunkt möchte ich noch einmal auf einen bekannten Fallstrick hinweisen, der sich besonders bei Einsteigern gerne einschleicht. PHP interpretiert jeden Ausdruck, der an das Sprachkonstrukt `if()` übergeben wird, als eine logische Aussage. Darum ist es besonders unangenehm, den Zuweisungsoperator = mit dem Vergleichsoperator == zu verwechseln.

```
<?
$erg = 1000 / 5;
//FEHLER !!!
if($erg = 200) {echo "1000 geteilt durch 5 ist 200!";}
if($erg = 250) {echo "1000 geteilt durch 5 ist 250!";}
?>
```

Listing 4.2: fehlerhafte Bedingung!

Ausgabe:

```
1000 geteilt durch 5 ist 200!
1000 geteilt durch 5 ist 250!
```

Diese Aussage ist immer TRUE, da eine Zuweisung immer wahr ist. Der PHP- Interpreter bemängelt diese Konstruktion nicht, er führt sie brav aus. Umso schwieriger ist es, den Fehler zu finden, wenn das Programm plötzlich Unsinn fabriziert.

4.2.3 if-else-Bedingungen

Die `if-else`-Bedingung wird eingesetzt, wenn man für den Anweisungsblock der `if`-Bedingung einen Alternativblock angeben möchte, der ausgeführt werden soll, wenn die Aussage FALSE ist.

Der zweite Block wird einfach über das Schlüsselwort an das Bedingungskonstrukt angehängt.

```
<?
$erg = 1000 / 5;
if($erg == 250) {echo "Das Ergebnis ist 250!";}
else {echo "Das Ergebnis ist NICHT 250!";}
?>
```

Listing 4.3: Die if-else-Bedingung

Ausgabe:

```
Das Ergebnis ist NICHT 250!
```

Die `else`-Anweisungen werden nur ausgeführt, wenn die `if`-Bedingung FALSE ist, ansonsten wird der erste Anweisungsblock ausgeführt. Einer der beiden Blöcke wird aber auf jeden Fall ausgeführt, da es eine Aussage wie MAYBE nicht gibt!

4.2.4 Verschachtelte Bedingungen

`if`-Bedingungen können beliebig miteinander verschachtelt werden. Das ermöglicht komplexe Bedingungskombinationen, die von beliebig vielen Vorbedingungen abhängig gemacht werden können. Der Nachteil einer solchen Konstruktion ist die mangelnde Übersicht der einzelnen Anweisungsblöcke. Welcher Block gehört zu welchem `if` oder `else`?

```
<?
$x = 20;
$y = 25;
if($y - $x == 5)
    { if($y % $x == 5)
        { if ($y / $x == 5)
            { echo "Erstaunlich!";}
        else { echo "weniger erstaunlich!"; }
        }
    else { echo "Warum auch nicht?";}
    }
else {echo "Anders kann es gar nicht sein!"; }
?>
```

Listing 4.4: Verschachtelte if-Bedingungen

Ausgabe:

```
weniger erstaunlich!
```

Es ist zwar möglich, den Programmcode durch Einschübe übersichtlicher zu gestalten, aber selbst das ist nicht immer der Weisheit letzter Schluss. Der PHP-Interpreter geht beim Kompilieren solcher Programme immer davon aus, dass ein `else`-Block ganz einfach zu dem letzten vorangegangenen `if`-Block gehört. Möchte man ganz sicher gehen, dann sollte man sein Programm nach dieser Logik konstruieren.

Der elseif-Zweig

Eine elegante Möglichkeit, ein wenig Ordnung in seine Programme zu bringen, ist der `elseif`-Zweig an einer Bedingung, der anstelle des `else`-Zweiges gesetzt werden kann. Er formuliert einfach eine weitere Bedingung für den Fall, dass in der vorangegangenen `if`-Abfrage ein FALSE das Ergebnis war.

```
<?
$x = 21;
$y = 3;
if ($x * $y < 50) {echo "Das Ergebnis ist kleiner 50!";}
elseif ($x * $y < 60) {echo "Das Ergebnis ist kleiner 0!";}
elseif ($x * $y < 70) {echo "Das Ergebnis ist kleiner 70!";}
elseif ($x * $y < 80) {echo "Das Ergebnis ist kleiner 80!";}
else {echo "Das Ergebnis ist zu groß!";}
?>
```

Listing 4.5: elseif-Block

Ausgabe:

```
Das Ergebnis ist kleiner 70!
```

Das Programm wäre natürlich auch ohne `elseif` möglich, allerdings würde es bei weitem nicht so strukturiert aussehen. So ist auf jeden Fall klar, welcher Teil des Programms wohin gehört. Die Ausgabe klärt außerdem, dass nach dem ersten TRUE einer Abfrage keine weiteren `elseif`-Blöcke mehr ausgeführt werden.

Es ist möglich anstelle von `elseif` auch `else if` zu schreiben. Technisch gesehen besteht kein Unterschied.

HTML-Code

Es ist nicht möglich, `if-` und `else`-Blöcke durch HTML-Code zu trennen. Der PHP-Code muss immer vollständig sein. Für Formatierungsanweisungen muss also der Befehl `echo` verwendet werden.

```
<b><font color="green">
<?
$x = 10;
$y = 20;
if ($x > $y)  {echo "x ist größer!";}
?>
</font></b>             //Fehler!!!
<i><font color="red">   //Fehler!!!
```

```
<?
else {echo "x ist kleiner!";}
?>
</font></i>
```

Listing 4.6: Fehler in der Formatierung!

Ausgabe:

```
Parse error: parse error in c:\Programme\Apache Group\Apache\htdocs/
gotoPHP4/listing41.php on line 10
```

4.2.5 Verknüpfung von Bedingungen

Im Kapitel über Operatoren haben wir bereits die Grundlagen für logische Verknüpfungen kennen gelernt. Das größte Anwendungsfeld für diese Verknüpfungen ist sicherlich bei der Formulierung von Bedingungen. Die Anwendung ist schnell erklärt.

Das folgende Beispiel zeigt die Kontrolle von mehreren Variablen, wie sie zum Beispiel bei einem Login-Vorgang tagtäglich zum Einsatz kommen. Die Variablen `$user` und `$passwd` werden in der Realität natürlich vom User eingegeben.

```
<?
$user = "admin";
$passwd = "geheim";
if ($user == "admin" and $passwd == "geheim")
     {echo "Zugang gestattet!";}
if ($user == "admin" and !($passwd == "geheim"))
     {echo "Das Passwort ist falsch!";}
if (!($user == "admin") and $passwd == "geheim")
     {echo "Der Username ist falsch!";}
if (!($user == "admin") and !($passwd == "geheim"))
     {echo "Alles ist falsch!";}
?>
```

Listing 4.7: Verknüpfte Bedingungen

Ausgabe:

```
Zugang gestattet!
```

Das Programm gibt je nach Wert der Variablen eine Statusmeldung aus, die den User darüber in Kenntnis setzt, ob seine Angaben richtig oder falsch waren. Experimentieren Sie ein wenig herum. Setzen Sie `$user` und

`$passwd` mal auf falsche Werte, um einen Eindruck von den jeweiligen Reaktionen zu bekommen.

Unter dem Blickpunkt der Sicherheit ist das Programm gewiss fragwürdig, denn es ist bestimmt keine gute Idee, einem Hacker mitzuteilen, welche seiner beiden Angaben richtig und welche falsch war. In einer realen Anwendung sollte man dies unterlassen.

4.2.6 Kurzschreibweisen

Wie viele andere Programmiersprachen bietet PHP auch eine Kurzschreibweise für die `if`-Bedingung an. Die Syntax ist einfach:

```
Bedingung ? Anweisungen bei TRUE: Anweisungen bei FALSE
```

Die Bedingung wird ohne `if` einfach mit einem Fragezeichen von den beiden Anweisungsblöcken getrennt. Der erste Anweisungsblock entspricht dem `if`-Block, der bei einer wahren Bedingung ausgeführt wird, während der zweite Block dem `else`-Teil entspricht, der bei einer falschen Bedingung zum Zuge kommt.

```
<?
$x = 12;
$y = 13;
$x < $y ? $a = 1: $a = 0;
echo $a;
?>
```

Listing 4.8: Alternative Schreibweise von if()-else

Ausgabe:

```
1
```

Der Einsatz dieser Schreibweise ist eine Geschmacksfrage, da er (meiner Meinung nach) den Quellcode recht schnell unleserlich macht. Gerade in großen Programmen, die man auch nach einigen Wochen oder Monaten verwenden und anpassen möchte, empfiehlt es sich deshalb, auf diese Schreibung zu verzichten.

4.2.7 Alternativen mit switch()-case

Die `switch()-case`-Anweisung ist eine spezielle Art der Bedingungsüberprüfung, denn sie erlaubt es, eine beliebige Anzahl von Alternativen auf

einmal zu testen. Müsste man diese Konstruktion mit `if`-Bedingungen »bauen«, dann würde man einige Abfragen hintereinander stellen müssen.

Mit `switch() case` sind wir in der Lage darauf zu verzichten und alle möglichen Alternativen über eine eigene `case`-Anweisung zu behandeln. PHP sucht dabei nach Übereinstimmungen mit dem Wert, der in der Kopfzeile von `switch()` angegeben wurde. Ist diese vorhanden, wird der entsprechende Code ausgeführt.

```
<?
$x = 20;
$y = 24;
switch ($y - $x)
{
         case 1: echo "Das Ergebnis ist 1!<br>"; break;
         case 2: echo "Das Ergebnis ist 2!<br>"; break;
         case 3: echo "Das Ergebnis ist 3!<br>"; break;
         case 4: echo "Das Ergebnis ist 4!<br>"; break;
         case 5: echo "Das Ergebnis ist 5!<br>"; break;
         case 6: echo "Das Ergebnis ist 6!<br>"; break;
         default: echo "Das Ergebnis ist kleiner 1 oder
größer 6!<br>"; break;
}
?>
```

Listing 4.9: switch()-case

Ausgabe:

```
Das Ergebnis ist 4!
```

Für jede Alternative muss über das Schlüsselwort `case` eine Lösung angeboten werden. Ist keine der angegebenen Alternativen korrekt, kann man über das Schlüsselwort `default` eine Standardmöglichkeit angeben, die immer dann genutzt wird, wenn es keine Übereinstimmung gab.

Die Anweisung `break` nach jeder Alternative veranlasst PHP die Abarbeitung der `switch()-case`-Konstruktion nach der ersten Übereinstimmung zu verlassen. Würde man diesen Befehl nicht setzen, betrachtete PHP jede weitere Lösung nach der ersten Übereinstimmung automatisch als TRUE.

```
<?
$x = 20;
$y = 24;
switch ($y - $x)
```

```
{
         case 1: echo "Das Ergebnis ist 1!<br>";
         case 2: echo "Das Ergebnis ist 2!<br>";
         case 3: echo "Das Ergebnis ist 3!<br>";
         case 4: echo "Das Ergebnis ist 4!<br>";
         case 5: echo "Das Ergebnis ist 5!<br>";
         case 6: echo "Das Ergebnis ist 6!<br>";
         default: echo "Das Ergebnis ist kleiner 1 oder größer
6!<br>";
}
?>
```

Listing 4.10: switch()-case ohne break

Ausgabe:

```
Das Ergebnis ist 4!
Das Ergebnis ist 5!
Das Ergebnis ist 6!
Das Ergebnis ist kleiner 1 oder größer 6!
```

Alle Anweisungen nach der ersten Übereinstimmung werden automatisch ausgeführt. `break` werden wir im Kapitel über Schleifen noch genauer kennen lernen.

Neben nummerischen Werten können auch einfache Strings verglichen werden. Allerdings ist es nicht möglich, komplette Bedingungen zu überprüfen, weil `switch()-case` nicht auf TRUE oder FALSE überprüft.

4.2.8 Übungen

1. Was ist ein Programmblock?
2. Wann wird ein Bedingungsblock ausgeführt?
3. Wann wird ein `else`-Block ausgeführt?
4. Was ist der `ifelse`-Zweig?
5. Wie werden Bedingungen verknüpft?
6. Wofür braucht man die `break`-Anweisung in der `switch-case`-Konstruktion?

4.3 Schleifen

Der Einsatz von Schleifen ist immer dann notwendig, wenn Anweisung mehrmals hintereinander abgearbeitet werden sollen. Die Anzahl der Schleifendurchgänge können von verschiedenen Bedingungen abhängig gemacht werden. PHP bietet mehrere Schleifenkonstruktionen an, die wir jetzt der Reihe nach besprechen werden.

4.3.1 Die while()-Schleife

Die einfachste Schleife ist die `while()`-Schleife, die wir bereits im Kapitel über Stringfunktionen kennen gelernt haben. Ihre Syntax folgt der C- und Java-Syntax und sieht so aus:

```
while (Bedingung) { Anweisungen }
```

Solange die Bedingung innerhalb der Klammern TRUE ist, werden die Anweisungen im nachfolgenden Block ausgeführt. Bei jedem Schleifendurchgang wird die Bedingung neu geprüft. Ist die Bedigung FALSE, wird der Anweisungsblock übersprungen und der PHP-Interpreter setzt seine Arbeit in der ersten Zeile unterhalb dieses Blocks fort.

```
<?
$i = 0;
echo "Die Schleife beginnt...<br>";
while($i <= 10)
{
       echo "i hat den Wert $i !<br>";
       $i++;
}
echo "... und endet wieder!";
?>
```

Listing 4.11: Die while()-Schleife

Ausgabe:

```
Die Schleife beginnt...
i hat den Wert 0 !
i hat den Wert 1 !
i hat den Wert 2 !
i hat den Wert 3 !
i hat den Wert 4 !
i hat den Wert 5 !
i hat den Wert 6 !
```

```
i hat den Wert 7 !
i hat den Wert 8 !
i hat den Wert 9 !
i hat den Wert 10 !
... und endet wieder!
```

Ist die Bedingung schon beim ersten Anlauf nicht erfüllt, dann wird die Schleife sofort übersprungen, ohne dass der Block auch nur einmal ausgeführt wird.

Typischerweise wird die `while()`-Schleife mit einer Zählvariablen benutzt, um die Abbruchbedingung zu definieren. Eine ähnliche Methode bietet die `for()`-Schleife.

4.3.2 Die do-while()-Scheife

Die `do-while()`-Schleife arbeitet nach demselben Prinzip wie die `while()`-Schleife aus dem letzten Kapitel. Der feine Unterschied liegt darin, dass bei der `do-while()`-Schleife die Abbruchbedingung erst nach dem Durchlaufen des Anweisungsblocks geprüft wird.

```
do { Anweisungen } while (Bedingung)
```

Das hat zur Folge, dass der Anweisungsblock unabhängig von der Bedingung auf jeden Fall einmal durchlaufen wird.

```
<?
$i = 1000;
do {
     echo "$i ist kleiner als 10!<br>";
     $i++;
} while($i < 10)
?>
```

Listing 4.12: Die do-while()-Schleife

Ausgabe:

```
1000 ist kleiner als 10!
```

Der Einsatz der `do-while()`-Schleife bietet sich dann an, wenn der Anweisungsblock auf jeden Fall einmal durchlaufen werden muss.

4.3.3 Endlosschleifen

Eine Gefahr und ein typischer Fehler beim Einsatz der `while()` bzw. `do-while()`-Schleife ist die Möglichkeit von Endlosschleifen. Diese treten immer dann auf, wenn die Abbruchbedingung der Schleife nicht erfüllt wird und sie sich bis in alle Ewigkeit fortsetzt.

```
<?
$i = 0;
echo "Die Schleife beginnt...<br>";
while($i <= 10)
{
       echo "i hat den Wert $i !<br>";
}
echo "... und endet wieder!";
?>
```

Listing 4.13: Eine Endlosschleife

Ausgabe:

```
Die Schleife beginnt...
i hat den Wert 0 !
i hat den Wert 0 !
i hat den Wert 0 !
i hat den Wert 0 !
i hat den Wert 0 !
....
```

Hier hat der Programmierer vergessen die Variable `$i` pro Schleifendurchgang zu inkrementieren. Der Fehler ist schnell gefunden. Gefährlich ist es, wenn die Abbruchbedingung durch Usereingaben formuliert wird, da gerade im Internet viel Unsinn fabriziert wird. Gibt der User anstelle einer Zahl einen Buchstaben ein, dann wird es schon problematisch.

Die meisten Interpreter sind so konfiguriert, dass ein Skript automatisch nach einer bestimmten Zeit abgebrochen wird, um so Endlosschleifen abzufangen. Trotzt allem sollte man darauf achten, dass man keine Endlosschleifen in seinen Programmen ermöglicht.

4.3.4 Die for()-Schleife

Die `for()`-Schleife ist die komplizierteste Schleifenkonstruktion, die PHP bietet. Sie unterscheidet sich von den übrigen Schleifen dadurch, dass sie die Steuervariablen für die Abbruchbedingung selbst definiert und manipuliert. Die Gefahr einer Endlosschleife ist hier also nicht so groß.

Der Kopf der Schleife besteht aus drei Teilen, die nach dem Schlüsselwort `for` in runden Klammern gesetzt werden. Danach folgt der Anweisungsblock in den bekannten geschweiften Klammern.

```
for (erste Zuweisung ; Bedingung ; dynamische Zuweisung) { Anweisungen }
```

Der erste Teil des Schleifenkopfes ist eine Zuweisung, die am Anfang des »Schleifenlebens« vorgenommen wird. In der Regel handelt es sich hier um eine Steuervariable, über die die Abbruchbedingung formuliert wird.

Nach dieser Zuweisung wird die eigentliche Abbruchbedingung festgelegt. Dieser Teil des Schleifenkopfes wird bei jedem neuen Durchlauf neu bewertet. Solange dieser Ausdruck TRUE ist, wird die Schleife fortgesetzt. Ist er FALSE, dann setzt der Interpreter seine Arbeit nach der Schleife fort. Die Abbruchbedingung sollte über die Steuervariable festgelegt sein, auch wenn dies keine Pflicht ist.

Der letzte Teil ist eine dynamische Zuweisung bzw. Manipulation der Steuervariablen. Typischerweise wird die Variable inkrementiert oder dekrementiert.

```
<?
for($i = 0; $i < 10; $i++)       //Schleifenkopf
{
     echo "i hat den Wert $i!<br>";
}
?>
```

Listing 4.14: Die for()-Schleife

Ausgabe:

```
i hat den Wert 0!
i hat den Wert 1!
i hat den Wert 2!
i hat den Wert 3!
i hat den Wert 4!
i hat den Wert 5!
```

```
i hat den Wert 6!
i hat den Wert 7!
i hat den Wert 8!
i hat den Wert 9!
```

Das Listing zählt die Variable `$i` hoch und bricht bei einem Wert von 10 ab. Der jeweilige Status wird über `echo` ausgegeben.

Komplexe for()-Schleifen

Die `for()`-Schleife ist sehr flexibel in ihrem Aufbau. Es ist ohne weiteres möglich, mehrere Zählvariablen zu initialisieren und gleichzeitig mehrere Abbruchbedingungen festzulegen. Auch die dynamische Zuweisung im dritten Teil der Schleife kann mehrere Anweisungen aufnehmen. Dabei muss es sich nicht zwangsläufig um Variablenmanipulation handeln, es können auch ganz normale Anweisungen sein.

```
<?
for($i = 1, $j = 100; $i < 10, $j % $i != 4; print("i hat den Wert
$i<br>"), $i++)
{...}
?>
```

Listing 4.15: Komplexe for()-Schleifen

Ausgabe:

```
i hat den Wert 1
i hat den Wert 2
i hat den Wert 3
i hat den Wert 4
i hat den Wert 5
```

Es ist also durchaus möglich, dass der komplette Anweisungsblock der `for()`-Schleife im Schleifenkopf untergebracht wird und der eigentliche Block leer bleibt. Inwieweit das sinnvoll ist, bleibt jedem selbst überlassen. Allerdings wird die Sache relativ schnell unübersichtlich.

4.3.5 Arrays und die foreach-Schleife

Seit der Version 4 unterstützt PHP einen weiteren Schleifentyp, der speziell für die Bearbeitung von Arrays geschaffen wurde. Perl-Programmierer kennen diese Schleife vielleicht schon und möchten sie bestimmt auch nicht missen, denn sie erleichtert die Arbeit mit Arrays ungemein.

Die `foreach()`-Schleife ermöglicht es, Arrayelemente der Reihe nach durchzugehen und nacheinander zu bearbeiten. Dabei wird dynamisch bei jedem Schleifendurchgang das nächste Arrayelement in eine Variable kopiert. Diese kann dann beliebig verwendet werden.

Die Syntax ist einfach:

```
foreach ($array as $element)
{ Anweisungen }
```

Die Variable `$element` steht im Anweisungsblock zur Verfügung und wird bei jedem Durchgang neu gesetzt. Der simpelste Weg alle Elemente nacheinander auszugeben würde also so aussehen:

```
<?
$liste = range(1,12);
foreach($liste as $var)
{
   echo $var . ". Platz<br>";
}
?>
```

Listing 4.16: Die foreach()-Schleife

Ausgabe:

```
1. Platz
2. Platz
3. Platz
4. Platz
5. Platz
6. Platz
7. Platz
8. Platz
9. Platz
10. Platz
11. Platz
12. Platz
```

So simpel und so genial! Für assoziative Arrays ändert sich die Syntax ein wenig und stellt sich so dar:

```
foreach ($array as $key => $element)
{ Anweisungen }
```

Bei jedem Schleifendurchgang werden die Variablen `$key` und `$element` neu gesetzt und stehen zur Verfügung. In der Praxis sieht das Ganze dann so aus:

```
<?
$liste = array(
"vorname" => "Dirk",
"nachname" => "Ammelburger",
"email" => "dirk@ammelburger.de",
"web" => "http://www.kulturbrand.de"
);
foreach($liste as $key => $element)
{
   echo $key ." => ". $element ."<br>";
}
?>
```

Listing 4.17: Die foreach()-Schleife und assoziative Arrays

Ausgabe:

```
vorname => Dirk
nachname => Ammelburger
email => dirk@ammelburger.de
web => http://www.kulturbrand.de
```

Das Programm gibt sowohl das Schlüsselwort als auch den dazugehörigen Wert aus.

4.3.6 break und continue

In Anlehnung an die C/C++-Syntax ist es in PHP möglich, unabhängig von den Schleifenvorgaben und Steuervariablen eine Schleife über Schlüsselwörter zu steuern. Die `break`-Anweisung haben wir bereits im Kapitel über die Fallunterscheidung mit `switch()-case` kennen gelernt.

In manchen Situationen ist es wünschenswert, dass eine Schleife vor dem eigentlichen Erfüllen der Abbruchbedingung beendet wird. Genau wie wir die Verarbeitung durch `switch()-case` mit `break` abgebrochen haben, kann auch eine Schleife unterbrochen werden.

Sobald der PHP-Interpreter auf das Schlüsselwort `break` stößt, wird die aktuelle Schleife abgebrochen und das Programm in der ersten Zeile nach der Schleife fortgesetzt.

```
<?
$i = 0;
while($i < 100)
{
   $i++;
```

```
   if($i >= 12) {break;}
   echo "i hat den Wert $i!<br>";
}
?>
```

Listing 4.18: Die Anweisung break

Ausgabe:

```
i hat den Wert 1!
i hat den Wert 2!
i hat den Wert 3!
i hat den Wert 4!
i hat den Wert 5!
i hat den Wert 6!
i hat den Wert 7!
i hat den Wert 8!
i hat den Wert 9!
i hat den Wert 10!
i hat den Wert 11!
```

Die Schleife in diesem Listing hätte eigentlich eine Lebensdauer von 100 Durchgängen. Aber aufgrund der `break`-Anweisung in der dritten Zeile wird die Schleife schon nach zwölf Durchgängen »abgewürgt«.

Die Anweisung `continue` hat eine recht ähnliche Aufgabe wie `break`. Mit ihr kann erreicht werden, dass der aktuelle Schleifendurchgang abgebrochen wird und der nächste Schleifendurchgang automatisch gestartet wird. Im Gegensatz zu `break` wird die Schleife aber fortgesetzt.

```
<?
$i = 0;
while($i < 100)
{
   $i++;
   if($i > 5 and $i < 95) {continue;}
   echo "i hat den Wert $i!<br>";
}
?>
```

Listing 4.19: Die Anweisung continue

Ausgabe:

```
i hat den Wert 1!
i hat den Wert 2!
i hat den Wert 3!
i hat den Wert 4!
```

```
i hat den Wert 5!
i hat den Wert 95!
i hat den Wert 96!
i hat den Wert 97!
i hat den Wert 98!
i hat den Wert 99!
i hat den Wert 100!
```

Das Listing sieht auf den ersten Blick aus wie das vorhergehende. Allerdings wurde die Anweisung `break` hier durch `continue` ersetzt und auch die Bedingung wurde um einen Punkt erweitert.

Das Programm durchläuft die Schleife auch diesmal komplett, auch wenn die Ausgabe etwas anderes suggeriert. Der Zahlenbereich zwischen 5 und 95 wird komplett weggelassen, da wir den Schleifendurchgang vor der Ausgabe über `continue` verlassen.

4.3.7 Übungen

1. Was gibt dieses Programm aus?

```
<?
$i = 10;
while($i > 0)
{
        echo "i hat den Wert $i !<br>";
        $i--
}
?>
```

2. Was ist der Unterschied zwischen der `while()`-Schleife und der `do-while()`-Schleife?
3. Was sind Endlosschleifen?
4. Was macht `die for()`-Schleife so flexibel?
5. Welchen Sinn hat die `foreach()`-Schleife?
6. Erläutern Sie den Unterschied zwischen `break` und `continue`.

4.4 Funktionen

Das Wort Funktion ist in den letzten Kapiteln schon einige Male gefallen. Meistens im Zusammenhang mit PHP-eigenen Funktionen wie `print()` oder `explode()`, die uns eine bestimmte Funktionalität zur Verfügung gestellt haben. Wir sind in unseren Programmen allerdings nicht nur auf

den Funktionsumfang, der uns von PHP geboten wird, beschränkt. Es ist möglich und auch sinnvoll, dass wir unsere eigenen Funktionen schreiben.

Allgemein gesprochen dient eine Funktion dazu, bestimmte Bereiche eines Programms abzugrenzen und in kleine mehrfach verwendbare Einheiten zu zerlegen. Wenn vor dem Programm klar ist, dass wir den Code für eine bestimmte Berechnung oder eine Datenbankverbindung oft brauchen werden, ist es sinnvoll, diesen in eine eigene Funktion zu schreiben anstatt ihn bei jeder Gelegenheit neu tippen zu müssen.

Eine Funktion setzt sich aus einem Funktionskopf und dem Anweisungsblock zusammen, der die eigentliche Funktionalität definiert. Der Funktionskopf besteht aus dem Schlüsselwort `function` gefolgt vom Funktionsnamen, der die Bezeichnung der Funktion festlegt. Nach dem Namen der Funktion folgen runde Klammern, die die erwarteten Parameter übernehmen. Diese Parameter stehen dann später als lokale Variablen innerhalb des Funktionsblocks zu Verfügung. Sollen keine Parameter übernommen werden, bleiben die Klammern einfach leer.

In der Regel endet eine Funktion mit einem Rückgabewert, der durch das Schlüsselwort `return` gekennzeichnet wird. Der Wert, der von `return` zurückgegeben wird, ergibt den eigentlichen Wert der Funktion. Soll kein Wert zurückgegeben werden, dann kann `return` auch weggelassen werden.

Das folgende Skript definiert eine Funktion mit dem Namen `summe()`, die die Summe der übergebenen Parameter berechnet und zurückgibt. Die Funktion hat keinen tieferen Sinn, sie demonstriert nur die Syntax einer eigenen Funktionsdefinition.

```
<?
function summe($a, $b)
{
    $c = $a + $b;
    return $c;
}
$erg = summe(4,5);
echo $erg;
?>
```

Listing 4.20: Die Funktion summe()

Ausgabe:

```
9
```

Die Funktion bekommt die beiden Werte 4 und 5 übergeben, die in den lokalen Variablen `$a` und `$b` gespeichert werden. Aus den beiden Werten wird die Variable `$c` berechnet, die zweifelsfrei den Wert 9 erhält. Dieser wird per `return` an das Hauptprogramm zurückgegeben, das die Variable `$erg` auf den Rückgabewert setzt. Zu guter Letzt wird das Ergebnis ausgegeben.

Wie das Listing zeigt, wird eine Funktion immer aus einer übergeordneten Programmebene heraus aufgerufen. Innerhalb einer Funktion dürfen allerdings weitere Funktionen aufgerufen werden, die ihr Ergebnis dann wieder an die Startfunktion zurückgeben.

4.4.1 Funktionen und lokale Variablen

In Kapitel 3.4.2 haben wir bereits über die Sichtbarkeit von Variablen gesprochen, ohne dabei näher auf die Funktionen einzugehen. Aus diesem Grund möchte ich das Thema noch einmal kurz wiederholen.

Per Definition sind alle Variablen innerhalb einer Funktion lokal vereinbart. Das heißt, dass Variablen in der Funktion nicht vom Hauptprogramm gelesen werden können. Gleichzeitig kann eine Funktion auch keine globale Variable überschreiben. Alle Variablen in einer Funktion werden automatisch bei deren Beendigung gelöscht.

Möchte man dies verhindern, muss man eine Funktionsvariable entweder über das Schlüsselwort `global` global definieren oder per `return` an das Hauptprogramm zurückgeben. Eine weitere Möglichkeit wäre das Schlüsselwort `static`, das PHP anweist, die betroffene Variable über die Lebenszeit der Funktion im Speicher zu behalten.

Alle Variablen, die einer Funktion als Parameter übergeben werden, sind lokale Variablen. Diese Variablen werden über das Prinzip *call by value* gesetzt. Das heißt, dass die Variablen, die den Wert möglicherweise übergeben haben, nichts mehr mit den Werten in der Funktion zu tun haben.

Möchte man das umgehen, dann ist es erforderlich, bei der Parameterübergabe das Prinzip *call by reference* zu erzwingen. Zu diesem Zweck übergeben wir der Funktion nicht den Wert einer Variablen, sondern deren Adresse. Damit erschaffen wir eine lokale Kopie derselben Variable in der Funktion.

```
<?
function testParameter($x)
{
        $x = 100;
}
$y = 20;
testParameter($y);      //call by value
echo $y;
echo "<br>";            //Zeilenumbruch
$z = 20;
testParameter(&$z);     //call by reference
echo $z;
?>
```

Listing 4.21: Parameterübergabe als Referenz

Ausgabe:

```
20
100
```

Im ersten Fall übergeben wir der Funktion `testParameter()` die Variable `$y` mit dem Wert 20. In diesem einfachen Fall wird nur der Wert der Variable übergeben, darum hat die Zuweisung in der Funktion keinerlei Auswirkung auf den Wert der globalen Variable `$y`.

Im zweiten Fall übergeben wir `testParameter()` die Variable `$z`, die ebenfalls dem Wert 20 hat. Das Besondere hierbei ist die Art der Übergabe an die Funktion, da wir diesmal eine Referenz übergeben. Die lokale Variable in der Funktion übernimmt die Adresse von `$z` und ist damit faktisch derselbe Wert. Dies beweist die Ausgabe von `$z` nach der Funktion, das den Wert 100 aus der Funktion übernommen hat.

4.4.2 Rekursive Funktionen

PHP erlaubt den Einsatz von rekursiven Funktionen. Rekursiv bedeutet, dass eine Funktion sich in ihrer Funktionalität selbst aufruft. Damit es keine Endlosschleife gibt, muss die Anzahl der Aufrufe allerdings begrenzt sein. Ein typisches Beispiel für eine rekursive Funktion ist die Berechnung der Fakultät.

Die mathematische Definition für die Fakultät ist:

```
n! = n * (n -1) * (n -2) * (n-3) *... * 2 * 1
```

n ist eine natürliche, positive Zahl über 0, die mit dem eigenen Wert multipliziert wird, um jeweils 1 reduziert. Das setzt sich so lange fort, bis 1 erreicht ist, da eine Multiplikation mit 0 bekanntlich 0 ergibt. Die Formel würde rekursiv definiert so aussehen:

```
n! = (n - 1)!
```

In PHP umgesetzt sieht die Funktionsdefinition mit einem Beispielprogramm so aus:

```
<?
function fakultaet($n)
{
    if($n < 1) {return 0;}
    //Fängt 0 und alle Minuszahlen ab und gibt 0 für
FALSE zurück!
    if(round($n)  != $n) {return 0;}
    //Fängt Kommazahlen ab und gibt 0 zurück!
    //round() rundet eine Kommazahl zur Ganzzahl.
    if ($n == 1) {return 1;}
    //Abbruchbedingung für die Rekursion
    echo "\$n hat den Wert $n<br>";
    //Ausgabe von $n
    $erg=$n * fakultaet($n -1);
    //rekursiver Funktionsaufruf
    echo "\$erg hat den Wert $erg<br>";
    //Ausgabe von $erg
    return $erg;
    //Rückgabe des Ergebnisses an die nächsthöhere Programmebene
}
echo "Das Ergebnis lautet: " . fakultaet(5);
?>
```

Listing 4.22: rekursive Berechnung der Fakultät

Ausgabe:

```
$n hat den Wert 5
$n hat den Wert 4
$n hat den Wert 3
$n hat den Wert 2
$erg hat den Wert 2
$erg hat den Wert 6
$erg hat den Wert 24
$erg hat den Wert 120
Das Ergebnis lautet: 120
```

Der wichtigste Punkt in einer rekursiven Funktion ist die Abbruchbedingung. Im vorliegenden Listing wird die Rekursion abgebrochen, wenn der Wert von `$n` gleich 1 ist. Um den Vorgang der Berechnung besser nachvollziehen zu können, wird bei jedem Funktionsaufruf der aktuelle Wert von `$n` und `$erg` ausgegeben. Das ist bei einer »richtigen« Funktion nicht nötig.

Anhand der Ausgabe wird deutlich, wie PHP die Funktionen der Reihe nach aufruft und (wenn `$n` den Wert 1 hat) der Reihe nach wieder beendet. Erst die letzte Funktion gibt das Ergebnis an das Hauptprogramm zurück, wo es über `echo` ausgegeben wird.

4.4.3 include und require

PHP stellt zwei Funktionen zur Verfügung, mit deren Hilfe Dateien in ein Skript eingebunden werden können. PHP liest dafür den Code aus der angegebenen Datei aus und setzt ihn an die Stelle der jeweiligen Funktion. Die Syntax der beiden Funktionen sieht so aus:

```
include(Datei);
require(Datei);
```

Der Dateiname muss dabei als String übergeben werden. Die Angabe erfolgt relativ zur Lage des Skriptes auf dem Server. Alternativ kann der Pfad auch direkt angegeben werden. Dabei ist allerdings das jeweilige Dateisystem des Servers zu beachten.

```
include("c:\php\scripte\script.php");
include("/home/usr/aburg/php/scripte/script.php");
```

Der Einsatz dieser Funktionen ist dann sinnvoll, wenn man in verschiedenen Dateien denselben Code einsetzen möchte. Es bietet sich zum Beispiel, an eine Datei anzulegen, in der alle Funktionen für Datenbankzugriffe gespeichert sind. Über einen einfachen `require()`, oder `include()`-Befehl stehen sie dann in jedem Programm zur Verfügung.

Unterschiede zwischen include und require

Augenscheinlich machen die beiden Funktionen genau dasselbe. Es besteht allerdings ein kleiner Unterschied, den man kennen sollte, auch wenn er im Alltag fast keine Rolle spielt.

Der Befehl `require()` wird vom PHP-Interpreter vor der eigentlichen Ausführung des Skriptes bearbeitet und das auch nur ein einziges Mal. Der Vorgang ist vergleichbar mit dem Preprozessor bei C und C++, der alle

#include-Anweisungen vor dem eigentlichen Kompilieren bearbeitet. Die Funktion wird durch den gewünschten Code ersetzt, der im Anschluss erst interpretiert wird. Diese Reihenfolge macht es für PHP unmöglich, den Befehl in einer Schleife zu interpretieren.

Im Gegensatz dazu wird die Funktion include() erst während der Laufzeit des Programms interpretiert. Der Code wird also erst eingesetzt, wenn der Interpreter über den entsprechenden Befehl stolpert.

include() ist also ein wenig flexibler und kann aufgrund dessen auch in Schleifen eingesetzt werden. Der Preis für diese Freiheit ist eine etwas langsamere Bearbeitung des Befehls. Darum sollte include() eigentlich nur genutzt werden, wenn require() den Anforderungen nicht mehr gewachsen ist.

```
<?
for($i=0; $i < 3; $i++)
{
include "script.txt";
}
require "script.txt";
?>
```

Listing 4.23: include() und require()

Ausgabe:

```
Hier sind wir in der Datei script.txt!
Hier sind wir in der Datei script.txt!
Hier sind wir in der Datei script.txt!
Hier sind wir in der Datei script.txt!
```

Neben dem eigentlichen Skript brauchen wir für dieses Programm noch eine Datei mit dem Namen »script.txt«. Sie enthält die Zeile:

```
<? echo "Hier sind wir in der Datei script.txt!<br>"; ?>
```

Über die Befehl include() und require() wird die Datei in das aktuelle Programm eingebunden. Die Ausgabe sind vier Zeilen, die über den Befehl echo aus der Textdatei ausgegeben werden.

4.4.4 Übungen

1. Welche Vorteile hat der Einsatz von Funktionen?
2. Was sind rekursive Funktionen?
3. Was ist der Unterschied zwischen `include` und `require`?

5 Wichtige Funktionen in PHP

Das folgende Kapitel beschäftigt sich mit wichtigen und oft benötigt PHP-Funktionen, die in keinem der anderen Kapitel Platz gefunden haben.

Im Einzelnen werden wir hier eine Reihe mathematischer Funktionen besprechen sowie Funktionen für die Behandlung von Datum und Zeit. Der letzte Teil beschäftigt sich mit allgemeinen Informationen über PHP und das aktuelle Skript.

5.1 Kapitelübersicht

- Mathematische Funktionen in PHP
- Datum und Zeit
- Informationsfunktionen

5.2 Mathematische Funktionen in PHP

Die Sprache PHP unterstützt eine Menge Funktionen, die die Arbeit mit Zahlen und mathematischen Problemen stark erleichtern. Die nachfolgende Tabelle beschreibt die wichtigsten Funktionen des PHP-Funktionsumfangs, so dass Sie in der Lage sein werden, alle möglichen mathematischen Rechnungen zu lösen.

Funktion	Beschreibung
`abs()`	der absolute Betrag wird zurückgegeben
`round()`	rundet eine Kommazahl
`ceil()`	rundet auf die nächste Ganzzahl auf
`floor()`	rundet auf die nächste Ganzzahl ab
`exp()`	Potenz zur Eulerschen Zahl
`log()`	natürlicher Logarithmus
`log10()`	Logarithmus zur Basis 10
`max()`	höchster Wert

Tabelle 5.1: Mathematische Funktionen von PHP

Funktion	Beschreibung
`min()`	niedrigster Wert
`sqrt()`	Quadratwurzel

Tabelle 5.1: Mathematische Funktionen von PHP (Forts.)

```
<?
echo abs(-20);
echo round(12.4);
echo ceil(17.9);
echo floor(13.7);
echo exp(2);
echo log(12);
echo log10(4);
echo max(12,13,15,14);
echo min(6,5,4,3,7,4);
echo sqrt(16);
?>
```

Listing 5.1: Mathematische Funktionen

Ausgabe:

```
20
12
18
13
7.3890560989307
2.484906649788
0.60205999132796
15
3
4
```

5.2.1 Winkelfunktionen

Unentbehrlich für die Berechnung von geometrischen Figuren und Kreisbahnen sind Winkelfunktionen, wie wir sie in der Schule kennen und schätzen gelernt haben. Ein weiteres Beispiel ist die Berechnung eines Kreisumfangs mit der Konstanten pi, die in PHP auch über eine Funktion zurückgegeben wird.

```
<?
$radius = 4.5;
$umfang = 2 * $radius * pi();
echo "Ein Kreis mit dem Radius $radius hat den Umfang $umfang!";
?>
```

Listing 5.2: Berechnung des Kreisumfangs

Ausgabe:

```
Ein Kreis mit dem Radius 4.5 hat den Umfang 28.2743338!
```

Neben der eher einfachen Funktion `pi()` hat PHP natürlich noch eine ganze Menge mehr zu bieten.

Funktion	Beschreibung
`sin()`	Sinus-Berechnung
`tan()`	Tangens-Berechnung
`cos()`	Cosinus-Berechnung
`asin()`	Arcus-Sinus
`acos()`	Arcus-Cosinus
`atan()`	Arcus-Tangens
`pi()`	gibt Pi zurück (Kreiszahl)

Tabelle 5.2: Mathematische Winkelfunktionen

```
<?
echo sin(90);
echo tan(120);
echo cos(243);
echo asin(0.43);
echo acos(0.7);
echo atan(3);
echo pi();
?>
```

Listing 5.3: Mathematische Winkelfunktionen

Ausgabe:

```
0.89399666360056
0.71312300978591
-0.45594227589512
```

```
0.44449277693582
0.79539883018414
1.2490457723983
3.1415926535898
```

5.2.2 Zufallszahlen

Zufallszahlen dienen oft als Grundlage für Spiele im Netz oder einfache Verschlüsselungsalgorithmen. PHP stellt eine Handvoll Funktionen zur Verfügung, die es erlauben, Zufallszahlen in PHP-Programmen zu erzeugen.

Die Grundlage für Zufallszahlen in fast jeder Programmiersprache ist die aktuelle Systemzeit des Rechners, auf dem das Programm läuft. Deshalb ist die Qualität der Zufallszahlen oft fragwürdig, vor allem wenn sie innerhalb sehr kurzer Abstände generiert werden. Oft kann man beobachten, dass sich bestimmte Zufallszahlen im Sekundenrhythmus verändern, was natürlich Rückschlüsse auf den Zufallsgenerator zulässt.

Funktion	Beschreibung
`rand()`	hibt eine zufällige Ganzzahl zwischen 0 und 32767
`getrandmax()`	gibt die höchste Zufallszahl zurück, die möglich ist
`srand()`	setzt den Standardwert der Zufallsberechnung fest

Tabelle 5.3: Zufallszahlen

```
<?
srand(199);
echo rand();
echo "<br>";
echo getrandmax();
echo "<br>";
?>
```

Listing 5.4: Zufallszahlen

Ausgabe:

```
688
32767
```

Alternativ für die Zufallszahlen, die direkt auf der Systemzeit basieren, stellt PHP eine weitere Methode zur Verfügung, mit der man Zufallszahlen berechnen kann. Diese Funktionen sind genauso aufgebaut wie die Funktionen, die wir gerade kennen gelernt haben, aber sie haben immer das Präfix `mt_`.

Funktion	Beschreibung
`mt_rand()`	verbesserte Zufallszahl
`mt_getrandmax()`	gibt die höchste Zufallszahl zurück, die möglich ist
`mt_srand()`	setzt den Standardwert der Zufallsberechnung fest

Tabelle 5.4: Verbesserte Zufallszahlen

```
<?
mt_srand(2);
echo mt_rand();
echo "<br>";
echo mt_getrandmax();
echo "<br>";
?>
```

Listing 5.5: Verbesserte Zufallszahlen

Ausgabe:

```
1309289693
2147483647
```

Die Ausgabe des Programms macht sofort deutlich, dass diese Funktionen einen weitaus größeren Spielraum für Zufallszahlen besitzen. Der höchste Wert wird mit 2147483647 angegeben. Der Preis dafür ist eine recht langsame Berechnung der Zahl.

5.2.3 Zahlensysteme

Wie die meisten Programmiersprachen unterstützt PHP neben dem bekannten Dezimalsystem einige andere Zahlensysteme. Die Grundlage für Berechnungen außerhalb des Dezimalsystems sind Umwandlungsfunktionen, die es erlauben, Zahlen von einem System ins andere zu konvertieren.

Das wichtigste und gleichzeitig das einfachste System ist das Binärsystem, auf dem die gesamte Rechnerlogik basiert. Es gibt nur die Ziffern 1 und 0,

die jede beliebige Zahl beschreiben können. Mit Binärcodes ist man in der Lage, komplizierte logische Zusammenhänge für eine Maschine sinnvoll darzustellen.

Möchten Sie mit Binärzahlen rechnen, dann brauchen Sie folgende Funktionen:

Funktion	Beschreibung
`bindec()`	Umwandlung einer Binärzahl in eine Dezimalzahl
`decbin()`	Umwandlung einer Dezimalzahl in eine Binärzahl

Tabelle 5.5: Binärfunktionen

```
<?
$x = 2469;
$y = decbin($x);
echo $y;
echo "<br>";
$z = bindec($y);
echo $z;
?>
```

Listing 5.6: Binärfunktionen

Ausgabe:

```
100110100101
2469
```

Das zweite Zahlensystem, das von PHP direkt unterstützt wird, ist das Hexadezimalsystem. Die nahe Verwandtschaft zum Binärsystem erlaubt es, mit dem Hexadezimalsystem Zahlen darzustellen, die als Binärzahlen zu lang und unhandlich wären.

Funktion	Beschreibung
`hexdec()`	Umwandlung einer Hexadezimalzahl in eine Dezimalzahl
`dechex()`	Umwandlung einer Dezimalzahl in eine Hexadezimalzahl

Tabelle 5.6: Hexadezimalfunktionen

```
<?
$x = 24691344;
$y = dechex($x);
echo $y;
echo "<br>";
$z = hexdec($y);
echo $z;
?>
```

Listing 5.7: Hexadezimalfunktionen

Ausgabe:

```
178c290
24691344
```

Um das Hexadezimalsystem darstellen zu können, werden für die Zahl nach der 9 die Buchstaben des Alphabetes verwendet. a steht für 10, b für 11 und so weiter. Diese Regelung wurde eingeführt, weil es keine Ziffernzeichen für diese Stellen gibt.

Das dritte und eher selten genutzte Zahlensystem, das PHP mit Konvertierungsfunktionen unterstützt, ist das Oktalsystem. Wie der Name schon sagt, basiert dieses System auf der Zahl 8, die jeweils den Stellenumbruch signalisiert.

Funktion	Beschreibung
`octdec()`	Umwandlung einer Oktalzahl in eine Dezimalzahl
`decoct()`	Umwandlung einer Dezimalzahl in eine Oktalzahl

Tabelle 5.7: Oktalfunktionen

```
<?
$x = 246913;
$y = decoct($x);
echo $y;
echo "<br>";
$z = octdec($y);
echo $z;
?>
```

Listing 5.8: Oktalfunktionen

Ausgabe:

```
742201
246913
```

5.2.4 Übungen

Ü

1. Wie wird die Konstante pi in PHP verwaltet?
2. Auf welchem Wert basieren die Zufallszahlen in PHP?
3. Welche Zahlensysteme unterstützt PHP?

5.3 Datum und Zeit

Eine Webseite im Internet steht und fällt mit ihrer Aktualität, das heißt mit dem Zeitpunkt, an dem sie erstellt wurde. Um diesen Ansprüchen gerecht zu werden, bietet PHP eine große Menge an Datums- und Zeitfunktionen, die es erlauben, den aktuellen Zeitpunkt festzustellen oder ein beliebiges Datum zu berechnen.

5.3.1 Der UNIX-Timestamp

Die Datums- und Zeitfunktionen von PHP basieren, wie in so vielen anderen Programmiersprachen, auf dem so genannten UNIX-Timestamp oder zu Deutsch Zeitstempel. Diese leicht mystische Einheit gibt die Anzahl der Sekunden zurück, die seit dem Begin der UNIX-Epoche, dem 1. Januar 1970, vergangen sind. Inzwischen hat sich dieses Format überall durchgesetzt, darum wird der UNIX-Timestamp auf jedem Rechnersystem verwendet. Einzige Ausnahme sind einige Macsysteme, die die Sekunden seit dem 1. Januar 1904 berechnen. Dieses Manko dürfte allerdings mit der Version MacOS X verschwunden sein, da das neue Kernel komplett auf UNIX basiert.

Um den UNIX-Timestamp zu erhalten, braucht man die Funktion `time()`, die den aktuellen Zeitstempel zurückgibt.

```
<?
$time = time();
echo $time;
?>
```

Listing 5.9: Der UNIX-Timestamp

Ausgabe:

```
1002879501
```

Möchte man den Zeitstempel eines beliebigen Datums berechnen, dann benötigt man die Funktion `mktime()`. Man übergibt ihr ein beliebiges Datum mit sekundengenauer Uhrzeit und erhält den dazugehörigen Zeitstempel.

```
<?
$time = time();
echo $time;
?>
<br><br>
<?
$time = mktime(12, 30, 20, 11, 30, 1978);
echo $time;
?>
```

Listing 5.10: Die Funktion mktime()

Ausgabe:

```
281273420
```

Das Listing berechnet den UNIX-Timestamp vom 30. Januar 1978 zur Uhrzeit 12 Uhr 30 und 20 Sekunden. Sollte man versuchen einen Zeitstempel vor dem 1. Januar 1970 zu berechnen, dann gibt die Funktion –1 zurück. Dasselbe gilt für jedes Datum nach dem 19. Januar 2038.

Der Grund dafür ist schnell erklärt. Der UNIX-Timestamp wird intern (auf jedem System) im C/C++-Datentyp *long integer* gespeichert. Da dieser Datentyp auf 4 Byte Speicher beschränkt ist, ist die größte Zahl, die er speichern kann 2147483647 (= $2^{31} - 1$).

Das klingt ziemlich groß, ist aber leider nicht groß genug. Wenn der UNIX-Timestamp den Zeitpunkt erreicht hat, der durch diese Zahl repräsentiert wird, dann gibt es einen Speicherüberlauf und das Datum ist nicht mehr darstellbar. Berechnet man diesen Zeitpunkt, dann kommt man auf den 19. Januar 2038 um 4 Uhr 14 und 7 Sekunden.

Experten vermuten, dass dieser Zeitpunkt ähnliche Befürchtungen hervorrufen wird, wie die legendäre Zeitumstellung auf das Jahr 2000, auch bekannt als Y2K-Bug.

Alle Angaben beziehen sich natürlich immer auf die aktuelle Zeitzone, in der der Rechner steht. Für die Umrechnung zur Greenwich-Zeit (GMT) stellt PHP spezielle Konvertierungsfunktionen zur Verfügung. Hier noch einmal die Funktionen im Überblick:

Funktion	Anwendung	Beschreibung
`time()`	`$stamp = time()`	gibt den aktuellen UNIX-Zeitstempel zurück
`mktime()`	`$stamp = mktime($stunde, $minute, $sekunde, $monat, $tag, $jahr)`	gibt den UNIX-Timestamp für ein spezielles Datum zurück
`microtime()`	`$string = microtime()`	gibt den UNIX-Timestamp als String in der Form Microsekunden, Sekunden zurück

Tabelle 5.8: Funktionen für den UNIX-Timestamp

5.3.2 Formatierte Zeitangaben

Zugegeben, der UNIX-Timestamp hat auf den ersten Blick nicht die Aussagekraft, die man sich wünscht. Darum ist es sinnvoll, die Zeitangaben für das menschliche Auge in ein gewohntes Format zu bringen. PHP bietet hier eine Reihe von Funktionen, die eigentlich keine Wünsche offen lassen.

PHP verfolgt dabei zwei Ansätze. Einmal gibt es die Möglichkeit, ein Datum in ein Array zu zerlegen, das für die weitere Bearbeitung im Programm geeignet ist. Der zweite Weg ist die direkte Formatierung eines Datums zu einem String, der bequem und einfach ausgegeben werden kann. Alle Funktionen haben den Timestamp als Grundlage.

Die Funktion date()

Die Funktion `date()` gibt die lokale Uhrzeit und/oder das Datum als String formatiert zurück. Die Formatierung kann beliebig festgelegt werden. Zur Steuerung dient ein Formatstring, der sich aus bestimmten Formatkürzeln zusammensetzt.

Der zweite Parameter ist optional ein Zeitstempel, der für die Berechnung verwendet werden soll. Wir dieser nicht übergeben, dann verwendet PHP den aktuellen Zeitstempel.

Der Formatierungsstring kann sich aus folgenden Kürzeln zusammensetzen, die von PHP durch den entsprechenden Wert ersetzt werden. Alle anderen Zeichen werden von PHP unverändert übernommen und können zur Formatierung genutzt werden.

Formatierungszeichen	Erklärung
a	am oder pm Angabe
A	AM oder PM Angabe
d	Tag des Monats
j	Tag des Monats ohne 0
t	Anzahl der Tage im Monat
z	Tage seit Jahresbeginn
D	Wochentagname abgekürzt
I	Wochentagname
w	Tag in der Woche
M	Monatsname abgekürzt
F	Monatsname
m	Monat im Jahr
n	Monat im Jahr ohne 0
H	Stunden im 24-Format
h	Stunden im 12-Format
G	Stunden im 24-Format ohne 0
g	Stunden im 12-Format ohne 0
i	Minuten
s	Sekunden
L	Schaltjahr (TRUE oder FALSE)
Y	Jahreszahl
y	Jahreszahl zweistellig
Z	Zeitabweichung vom GTM

Tabelle 5.9: date() Formatierungszeichen

```
<?
echo date("d.M Y H:i:s");
echo "<br>";
echo date("d.M Y H:i:s", 2147483647);
?>
```

Listing 5.11: Die Funktion date()

Ausgabe:

```
12.Oct 2001 12:34:13
19.Jan 2038 04:14:07
```

Durch die flexible Handhabung der Ausgabe ist die Funktion `date` hervorragend geeignet, um webseitentaugliche Ausgaben von Datum und Zeit zu machen.

Das Datum als Array

Möchte man ein Datum als Array in einem Programm für Berechnungen verwenden, dann kann man die Funktion `getdate()` verwenden, die ein assoziatives Array mit dem Datum zurückgibt. Genau wie bei `date()` ist die Angabe eines Timestamps optional. Wird er weggelassen, verwendet `getdate()` den aktuellen Zeitstempel.

```
<?
$arr = getdate();
foreach ($arr as $key => $element)
{
echo "$key: $element";
echo "<br>";
}
?>
```

Listing 5.12: Die Funktion getdate()

Ausgabe:

```
seconds: 7
minutes: 52
hours: 12
mday: 12
wday: 5
mon: 10
year: 2001
yday: 284
weekday: Friday
month: October
0: 1002883927
```

Das Listing sollte selbsterklärend sein. Die Schlüsselwörter des Arrays sind intuitiv zu verwenden, man sollte allerdings daran denken, dass PHP mit englischen Ausdrücken arbeitet. Das letzte Element ist immer der Timestamp.

Die zweite Zeitfunktion, die mit Arrays arbeitet, ist `gettimeofday()`. Dieser sehr sprechende Name sagt eigentlich schon, worum es geht: Die Funktion gibt die aktuelle Zeit des heutigen Tages als Timestamp zurück. Zusätzlich erhält man die vergangenen Millisekunden des Tages sowie eine Angabe über Sommer- oder Winterzeit (TRUE oder FALSE).

```
<?
$arr = gettimeofday();
foreach ($arr as $key => $element)
{
echo "$key: $element";
echo "<br>";
}
?>
```

Listing 5.13: Die Funktion gettimeofday()

Ausgabe:

```
sec: 1002884148
usec: 51328647
minuteswest: -3600
dsttime: 1
```

Der Schlüssel `usec` sind die Millisekunden, während `minuteswest` die Zeitverschiebung von Greenwich in Sekunden angibt. Das letzte Element informiert darüber, ob gerade die Sommerzeit herrscht oder nicht. Eine 1 steht für TRUE und verrät uns, dass es draußen noch warm sein sollte.

Funktion	Beschreibung
`getdate()`	gibt ein Array mit Zeitangaben zurück
`gettimeofday()`	gibt ein Array mit genaueren Zeitangaben zum aktuellen Tag zurück

Tabelle 5.10: Array-Zeitfunktionen

5.3.3 Die Funktion checkdate()

Die Funktion `checkdate()` ist eine einfache Möglichkeit ein beliebiges Datum auf Gültigkeit zu überprüfen. Das kann auf der einen Seite nötig sein, um eine Userangabe zu kontrollieren (Wann wurden Sie geboren?) oder um festzustellen, ob ein Jahr ein Schaltjahr ist.

Die Parameter für diese Funktion beschränken sich auf das Datum, die als drei Integerwerte übergeben werden. Die Funktion gibt entweder TRUE (für gültig) oder FALSE (für nicht gültig zurück).

```
<?
$tag = 29;
$monat = 2;
$jahr = 2002;
while($jahr <= 2020)
{
if (checkdate($monat, $tag, $jahr))
{echo "Das Jahr $jahr ist ein Schaltjahr!<br>";}
$jahr++;
}
?>
```

Listing 5.14: Die Funktion checkdate()

Ausgabe:

```
Das Jahr 2004 ist ein Schaltjahr!
Das Jahr 2008 ist ein Schaltjahr!
Das Jahr 2012 ist ein Schaltjahr!
Das Jahr 2016 ist ein Schaltjahr!
Das Jahr 2020 ist ein Schaltjahr!
```

5.3.4 Die GMT-Zeitzone

Wie ich bereits angedeutet habe, beziehen sich alle Zeitfunktionen immer auf die lokale Zeitzone, für die der Rechner konfiguriert wurde. Möchte man das umgehen, muss man die GMT-Funktionen von PHP benutzen, die es erlauben, die Zeit in den Breitengraden von Greenwich zu berechnen.

Im Prinzip reichen zwei Funktionen aus, um diesem Anspruch gerecht zu werden. Einmal die Funktion `gmdate()`, die dieselbe Aufgabe hat wie `date()`, nur dass sie eben eine GMT-Zeit zurückgibt.

Genauso verhält es sich mit `gmmkdate()`. Diese Funktion berechnet genau wie `mkdate()` den UNIX-Timestamp zu einem Datum, mit dem Unterschied, dass auch hier die Greenwichzeit berücksichtigt wird.

```
<?
$gmtime = gmmktime(12, 30, 20, 11, 30, 1978);
echo $gmtime;
$gmstr = gmdate("d.M Y H:i:s", $gmtime);
$str = date("d.M Y H:i:s", $gmtime);
echo "<br>";
echo $gmstr;
echo "<br>";
echo $str;
?>
```

Listing 5.15: GMT-Zeitfunktionen

Ausgabe:

```
281273420
30.Nov 1978 11:30:20
30.Nov 1978 12:30:20
```

Die Ausgabe zeigt deutlich den Zeitunterschied zwischen der Greenwich-Zeit und der lokalen Zeit in good old Germany. Es ist sicher empfehlenswert, eine internationale Webseite mit einer GMT-Zeit auszurüsten, um eventuelle Missverständnisse zu vermeiden.

5.3.5 Übungen

1. Was ist der UNIX-Timestamp?
2. Was gibt dieses Skript aus?

   ```
   <?
   echo date("d.M Y H:i:s");
   echo "<br>";
   ?>
   ```

3. Was gibt die Funktion `getdate()` zurück?

5.4 Informationsfunktionen

PHP stellt ein paar Funktionen zur Verfügung, die es erlauben, Informationen über den benutzten PHP-Interpreter und das aktuelle Skript zu erhalten. Diese Methoden eignen sich nicht unbedingt für ein Anwen-

dungsprogramm, aber sie können dem Programmierer für seine Arbeit nützliche Hinweise liefern.

Die einfachste Funktion, die gleichzeitig den beeindruckendsten Output hat, ist `phpinfo()`. Sie wird ohne Parameter einfach aufgerufen und gibt eine komplette HTML-Seite mit allen relevanten Informationen über PHP und den Server aus.

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

Listing 5.16: phpinfo()

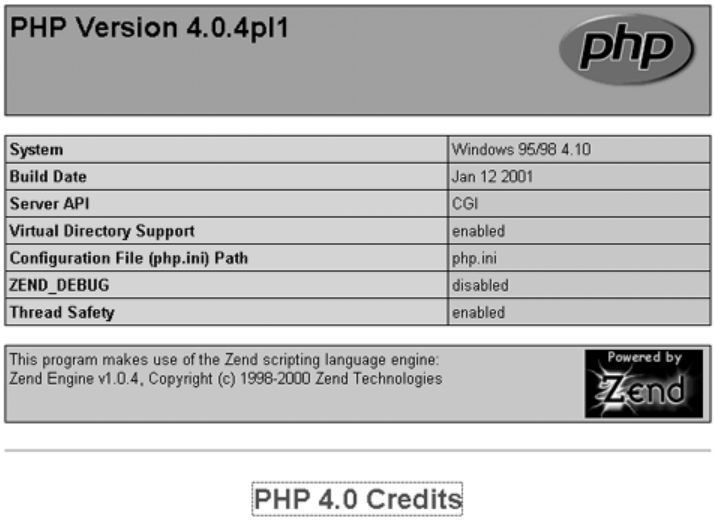


Abbildung 5.1: Ausgabe phpinfo()

Die Ausgabe erstreckt sich etwa über acht Seiten und liefert neben der aktuellen Konfiguration auch Informationen über Umgebungsvariablen und Angaben über das benutzte Betriebssystem. Wer will, kann sich mit einem Klick auf *PHP 4 Credits* auch noch über die Macher dieser Sprache informieren.

Wem es etwas kürzer auch genügt, der kann es mit der Funktion `phpversion()` versuchen. Sie gibt einfach die genaue Versionsbezeichnung des PHP-Interpreters zurück.

```
<?
echo phpversion();
?>
```

Listing 5.17: phpversion()

Ausgabe:

```
4.0.4pl1
```

Die Funktion verrät mir, dass es vielleicht Zeit wird, meine PHP-Version zu aktualisieren, da es bereits Version 4.0.5 gibt.

Die letzte recht nützliche Funktion ist `getlastmod()`. Sie gibt den Zeitpunkt der letzten Änderung an dieser Datei zurück, und zwar in der Form eines UNIX-Timestamp. Möchte man diese Ausgabe in ein leserliches Format bringen, braucht man die Funktion `date()`, die wir im letzten Kapitel kennen gelernt haben.

```
<?
$x = getlastmod();
echo $x;
echo "<br>";
echo date("d.M Y H:i", $x);
?>
```

Listing 5.18: getlastmod();

Ausgabe:

```
1002836658
11.Oct 2001 23:44
```

Da ich die Datei gerade eben erstellt habe, gibt `getlastmod()` das aktuelle Datum mit Uhrzeit zurück. Es ist also der 11. Oktober 2001, kurz vor Mitternacht. Wenn Sie schon immer mal wissen wollten, wann Autoren ihre Brötchen verdienen, dann haben Sie hier eine ehrliche Antwort. PHP ist unbestechlich ...

5.4.1 Übungen

Ü

1. Welchen Sinn haben PHP-Informationsfunktionen?
2. Was macht die Funktion `phpinfo()`?
3. Wie würden Sie feststellen, wann Sie zuletzt an einem Skript gearbeitet haben?

6 Fehlersuche

Die Fehlersuche in Skripten ist ein langwieriger und aufwändiger Prozess, der oft genauso viel Zeit in Anspruch nimmt wie das Schreiben des Programms selbst.

Zyniker reden oft von der so genannten 80:80-Regel, die maßgeblich für die Realisation eines Programms sein soll. Man braucht 80% der veranschlagten Zeit für die Planung und Erstellung des Programms und dann noch einmal 80% für die anschließende Fehlersuche.

Inwieweit diese Aussage wahr ist, muss jeder für sich selbst entscheiden. Dieses Kapitel soll dazu beitragen, die Zeit für die Suche nach Fehlern möglichst effektiv zu gestalten, damit man mehr Anteil an den wirklich schönen Dingen im Leben hat.

Der englische Ausdruck für Fehlersuche ist »Debugging« und wird Ihnen sicher schon begegnet sein. Die Geschichte behauptet, dass dieser Ausdruck von wirklichen Wanzen (engl.: Bug) herrührt, die sich in den großen Relais der ersten Rechner verfangen und so Fehler provoziert hatten.

Die meisten Programmierfehler können heute allerdings nicht unglücklichen Insekten angelastet werden, sondern gehen meistens auf unser eigenes Konto. Darum ist es heute umso wichtiger, gerade bei komplexen Programmen die Möglichkeiten der Debugging Tools »seiner« Sprache zu kennen.

6.1 Kapitelübersicht

- PHP-Fehlersystematik
- Semantische Fehler
- Laufzeitfehler

6.2 PHP-Fehlersystematik

PHP unterscheidet zwischen vier Klassen von Ereignissen, die in einem Programm auftreten können. Jedem Ereignis wird ein so genannter Bitwert zugeordnet, der es erlaubt den Ereignistyp festzulegen.

Ereignis	Beschreibung
Fehler (1)	syntaktischer Fehler im Skript
Warnung (2)	PHP gibt eine Warnung aus (z.B.: Variable nicht gesetzt!), das Programm läuft aber weiter.
Parserfehler (3)	Der PHP-Parser kann Teile des Programms nicht lesen.
Nachricht(4)	Meldung über einen bestimmten Zustand

Tabelle 6.1: Ereignisse in PHP

Anhand der Summe der Bitwerte aller Ereignisse wird in der PHP-Konfigurationsdatei `php.ini` ein Schwellenwert festgelegt, der sagt, wann eine Meldung im Browser erscheinen soll. Der Standardwert ist 7, das heißt es werden alle Ereignisse gemeldet, bis auf einfache Nachrichten, die den Programmverlauf nicht stören.

Möchte man dieses Verhalten ändern, muss in der Datei php.ini die Zeile

```
error_reporting = WERT
```

editiert werden. Der jeweilige Wert ergibt sich aus der Summe der Bitwerte, die ausgegeben werden sollen.

Anhand der Meldung im Browser ist es kein Problem, den Fehler zu lokalisieren und zu korrigieren. Der PHP-Interpreter ist dazu noch so nett und macht eine exakte Zeilenangabe, wo er den Fehler entdeckt hat.

```
<?
while ($x < 10)
{
echo "Hallo PHP!<br>";
$x++;           //FEHLER: geschweifte Klammer fehlt.
?>
```

Listing 6.1: Fehlerhaftes Skript I

Ausgabe:

```
Parse error: parse error in c:\Programme\Apache Group\Apache\htdocs/
gotoPHP4/listing55.php on line 6
```

Im Browser erscheint die Meldung, dass in Zeile 6 ein Parse-Error vorliegt, der dank dieser Angabe auch schnell gefunden wird. Der Programmierer hat schlicht vergessen, den Schleifenblock mit einer Klammer abzuschließen.

Man sollte den Zeilenangaben des Interpreters allerdings nicht blind vertrauen, da Computerlogik nicht immer der gewünschten Programmlogik gleicht. Der PHP-Interpreter sucht das fehlende Zeichen so lange, bis er entweder am Ende des Programms ist oder es syntaktisch keine andere Möglichkeit mehr gibt. An dieser Stelle wird dann die Fehlerausgabe gemacht, oft leider viel zu spät.

```
<?
echo "Hier beginnt die Schleife!<br>";
while($x < 10)
{
echo "Hallo PHP";
$x++;
        //Hier sollte die Schleife abgeschlossen werden!
echo "Hier endet die Schleife!";
?>
```

Listing 6.2: Fehlerhaftes Skript II

Ausgabe:

```
Parse error: parse error in c:\Programme\Apache Group\Apache\htdocs/
gotoPHP4/listing55.php on line 9
```

Auch hier wieder eine Fehlermeldung, die allerdings Verwirrung stiften kann. PHP gibt an, dass der Error in Zeile 9 zu finden ist, obwohl der logische Programmablauf die Klammer nach Zeile 6 verlangt.

Die korrekte Interpretation der Fehlermeldung wäre also, dass der Fehler bei Zeile 9 aufgefallen ist und irgendwo darüber liegen muss. Mit dem Wissen im Hinterkopf sollten syntaktische Fehler kein Problem mehr sein.

6.3 Semantische Fehler

Formale Fehler in einem Programm sind in der Regel kein großes Problem, weil sie automatisch vom PHP-Interpreter erkannt und gemeldet werden. Ungleich schwieriger wird es allerdings, wenn es sich um inhaltliche Fehler handelt, die zwar syntaktisch korrekt sind, aber logische Schwächen haben.

PHP findet solche Fehler natürlich nicht, da der Interpreter keine Annahme über die Funktion des Programms treffen kann. Sollte Ihr Programm also merkwürdige Ergebnisse liefern oder schlicht nicht das tun, was Sie erwarten, dann müssen Sie einen anderen Weg gehen.

Die einzige Chance, logischen Fehlern auf die Spur zu kommen, besteht darin, die Zeilen des »verdächtigen« Skripts per Hand nachzuvollziehen. Es erweist sich oft als hilfreich, die betroffenen Variablen während der Laufzeit zu überprüfen oder auszugeben. Geeignete Funktionen hierzu sind Abfragen zum aktuellen Variablenzustand.

Funktion	Beschreibung
`echo $a, $b, $c;`	gibt den Wert mehrerer Variablen aus
`print($a);`	gibt den Wert einer Variable aus
`empty()`	überprüft, ob die Variable leer (Standardwert) ist
`gettype()`	ermittelt den Variablentyp
`isset()`	überprüft, ob die Variable gesetzt ist

Tabelle 6.2: Prüffunktionen für die Fehlersuche

Eine weitere Möglichkeit ist das systematische Auskommentieren von Codezeilen, die den Fehler verursacht haben könnten. Ein kleines Beispiel zeigt das Vorgehen bei einer typischen Fehlersuche.

```
<?
function fakultaet($n)
{
    if($n < 1) {return 0;}
    if(round($n)  != $n) {return 0;}
    if ($n = 1) {return 1;}
    $erg=$n * fakultaet($n -1);
    return $erg;
}
echo fakultaet(5);
?>
```

Listing 6.3: Skript mit semantischem Fehler!

Als Grundlage haben wir hier die Funktion für die Fakultätsberechnung aus einem der vorhergehenden Kapitel. Allerdings hat sich ein Fehler eingeschlichen, da das Skript unabhängig vom Übergabeparameter immer 1 zurückgibt.

Um den Fehler zu finden durchsetzten wir die komplette Funktion mit Ausgaben, die es erlauben, den jeweiligen Zustand von `$n` zu erkennen. Zusätzlich nummerieren wir die Positionen, damit wir wissen, an welcher Stelle der Interpreter sich befindet.

```
<?
function fakultaet($n)
{
    if($n < 1) {return 0;}
    echo "Position1: $n<br>";
    if(round($n)  != $n) {return 0;}
    echo "Position2: $n<br>";
    if ($n = 1) {return 1;}
    echo "Position3: $n<br>";
    $erg=$n * fakultaet($n -1);
    echo "Position4: $n<br>";
    return $erg;
}
echo fakultaet(5);
?>
```

Listing 6.4: Skript mit semantischem Fehler und Ausgabe

Ausgabe:

```
Position1: 5
Position2: 5
1
```

Die Ausgabe verrät uns, dass das Skript bereits nach der zweiten Position abbricht und 1 zurückgibt. Das schränkt den Fehlerbereich stark ein und ein kurzer Blick an die entsprechende Stelle macht den Grund auch deutlich.

```
if ($n = 1) {return 1;}
```

Anstelle des Vergleichsoperators == wurde eine Zuweisung in der `if`-Bedingung vorgenommen, die immer TRUE zurückliefert. Es handelt sich also um einen einfachen Tippfehler. Wird die Stelle korrigiert, dann läuft das Skript wie gewünscht.

6.4 Laufzeitfehler

Ein weiterer heikler Punkt in einem Programm sind so genannte Laufzeitfehler, die erst während der Abarbeitung des Skripts auftreten. Typische Beispiele für solche Situationen sind Zugriffe auf fremde Ressourcen, wie zum Beispiel Dateien oder Datenbanken.

Das Skript kann vorher nicht »wissen«, ob die entsprechende Datei oder Datenbank an der angegebenen Stelle liegt oder nicht. Die einzige Möglichkeit auf eine solche Situation zu reagieren, ist eine Alternative bereitzuhalten.

Um unschöne Fehlermeldungen zu erklären, bietet PHP die Möglichkeit, das Skript sauber zu beenden und sogar eine eigene Meldung auf dem Browser auszugeben. Die Funktionen hierzu lauten `exit` und `die()`.

Funktion	Beschreibung
`exit`	beendet das Programm und räumt den Speicher
`die($text)`	genau wie exit, kann aber eine Nachricht ausgeben

Tabelle 6.3: Funktionen zur Behandlung von Laufzeitfehlern

Im folgenden Beispiel wird versucht eine nicht existierende Variable per `include()` zu öffnen. Der Aufruf wird von einer if-Bedingung umschlossen.

```
<?
echo "Dies ist ein Programm mit Laufzeitfehler:<br>";
if (!(include("prog.txt")))
{die ("Datei nicht gefunden! Fehler in Zeile ". __LINE__);}
?>
```

Listing 6.5: Fehlerbehandlung mit die()

Ausgabe:

```
Dies ist ein Programm mit Laufzeitfehler:
Datei nicht gefunden! Fehler in Zeile 4
```

Wird die Datei `prog.txt` nicht gefunden, dann gibt die Funktion `include()` FALSE zurück, die durch den !-Operator in ein TRUE umgewandelt wird. Das ist der Auslöser für den Programmabbruch durch `die()` mit einer entsprechenden Fehlermeldung.

Die Anweisung `exit` funktioniert auf demselben Weg, allerdings ohne die Ausgabe einer entsprechenden Meldung.

6.4.1 Übungen

Ü

1. Nach welchem System arbeitet die PHP-Fehlersystematik?
2. Was ist das Problem der Zeilenangabe bei Fehlermeldungen?
3. Was sind semantische Fehler?
4. Welche Möglichkeiten gibt es, semantische Fehler zu finden?
5. Was sind Laufzeitfehler?
6. Wie kann man Laufzeitfehlern begegnen?

Teil II – Going professional!

7 Interaktive Webseiten mit PHP

Wie wir bereits in der Einleitung festgestellt haben, ist das Ziel interaktive Webseiten zu erschaffen, die es erlauben, mit dem User zu kommunizieren und auf seine Bedürfnisse einzugehen.

In der Regel sind Webseiten im Netz nicht interaktiv. Es werden lediglich Informationen präsentiert, die manuell erneuert werden müssen. Solche Webseiten eignen sich nicht sonderlich, um mit einem Surfer in Kontakt zu treten und mehr über seine Bedürfnisse zu erfahren.

Zeitgemäße Webseiten dagegen erlauben heutzutage vielfältige Interaktion mit dem Benutzer und den Austausch von Informationen zwischen dem Browser (Client) und dem Server. Die übermittelten Informationen werden mittels PHP-Skripte ausgewertet und bearbeitet, um eine personalisierte Antwortseite generieren zu können.

Das wohl bekannteste Beispiel sind so genannte Shops, über die im Internet eingekauft werden kann. Jeder User erhält einen Einkaufskorb, in dem die gewählten Artikel gespeichert werden. Diese Information wird auf dem Server abgelegt oder von Seite zu Seite übergeben, damit am Ende der Shoppingtour erfolgreich bestellt werden kann.

7.1 Kapitelübersicht

- Formulare – Datenübergabe auf dem einfachsten Wege
- Cookies – Daten lokal speichern
- Sessions – Wie man einen User wiedererkennt

7.2 Formulare

Die Grundlage für interaktive Webseiten im Internet sind Formulare. Sie stellen das am häufigsten verwendete Medium dar, um im Internet Daten zu speichern und weiterzugeben. Es gibt zwar noch einige andere Möglichkeiten Daten über die Lebenszeit einer HTML-Seite hinaus zu speichern (die wir alle noch kennen lernen werden), aber Formulare sind der einfachste und effektivste Weg.

Mit ein Grund für den großen Erfolg von PHP stellt die wirklich einfache Handhabung von Formulardaten in Skripten dar. Wird ein PHP-Skript von einem Formular aufgerufen, stehen alle übergebenen Daten automatisch als Variable zur Verfügung, ohne das der Programmierer etwas dafür tun muss.

Andere Sprachen wie Perl oder sogar C/C++ (ja, auch mit C/C++ kann man Webserver programmieren) machen die Sache deutlich komplizierter. Hier muss der Übergabestring des HTTP-Request »von Hand« zerlegt und decodiert werden. Das ist bei PHP nicht nötig, da der Interpreter das automatisch erledigt.

7.3 Formulare mit HTML

Die Sprache HTML stellt spezielle Elemente bereit, mit denen ein Programmierer unkompliziert Formulare erstellen kann. Ein Formular erlaubt dem User Eingabefelder im Browser auszufüllen, Buttons anzuklicken, Elemente auszuwählen und diese Daten an den Server zu schicken.

Der Webserver leitet diese Eingaben an das aufgerufene PHP-Skript weiter, welches sie auswertet. Abhängig von den übermittelten Daten wird eine Antwortseite generiert und ausgegeben.

Ein Formular wird in einer HTML-Seite durch den `<FORM>`-Tag definiert. Er beschreibt den Anfang und das Ende eines Formulars und umschließt alle weiteren Formular-Tags. Der Formular-Tag hat drei verschiedene Attribute, die das Verhalten des Formulars steuern.

Attribut	Beschreibung
`ACTION`	Gibt die Zieladresse an, an die der Formularinhalt geschickt werden soll.
`METHOD`	Beschreibt die Methode, die zur Übertragung verwendet werden soll. Es stehen GET oder POST zur Verfügung.
`NAME`	Legt den Namen des Formulars fest. Wird in der Regel nur für JavaScript-Anwendungen verwendet.

Tabelle 7.1: Formular-Attribute

Eine einfache Formulardefinition könnte in der Praxis so aussehen:

```
<FORM ACTION="request.php" METHOD="POST" NAME="myForm">
...
</FORM>
```

Listing 7.1: HTML-Formular

Sobald das Formular abgeschickt wird, sendet der Browser die Daten an das Skript *request.php* im selben Verzeichnis auf demselben Server. Selbstverständlich kann man auch eine komplette URL angeben. Als Übertragungsmethode wird POST verwendet, das bedeutet, die Daten werden als Umgebungsvariablen auf dem Server gespeichert. Die GET-Methode würde die Daten in der URL speichern.

7.3.1 Der INPUT-Tag

Der `INPUT`-Tag ist das vielseitigste Formularelement und beschreibt je nach Typ unterschiedliche Eingabeobjekte. Er hat in einer HTML-Seite nur innerhalb des Formular-Tags Gültigkeit und dient dazu, Steuerelemente zu definieren.

Über das Attribut `TYP` wird die Art des Steuerelements festgelegt. Dazu bedient man sich einer Reihe von Konstanten, von denen jedes ein typisches Windowselement beschreibt. Folgende Konstanten stehen zur Verfügung:

TYP	Beschreibung
`text`	definiert ein einfaches Eingabefeld
`password`	wie `text`, aber die Eingabe erfolgt versteckt (Passwortfeld)
`hidden`	ein unsichtbares Steuerelement. Dient dazu, Daten versteckt zu übermitteln
`radio`	definiert einen Radiobutton
`checkbox`	definiert eine Checkbox
`button`	definiert einen Button
`submit`	definiert einen Button, der bei einem Klick das Formular abschickt
`reset`	definiert einen Button, der bei einem Klick das Formular löscht

Tabelle 7.2: Verschiedene Steuerelemente

Je nach Typ unterstützt der INPUT-Tag noch ein paar weitere Attribute, die das Verhalten des Objekts noch genauer definieren. Nicht immer sind alle Attribute sinnvoll und werden bei Bedarf einfach ignoriert.

Attribut	Beschreibung
TYP	legt den Typ des Steuerelements fest
NAME	legt den Namen des Steuerelements fest
VALUE	definiert einen Wert für das Steuerelement
CHECKED	Radiobuttons und Checkboxes sind markiert
MAXLENGTH	legt die maximale Länge der Eingabe fest
SIZE	legt die Breite eines Eingabefensters fest

Tabelle 7.3: Der INPUT-Tag

Im Gegensatz zur Situation beim FORM-Tag ist der Name eines Steuerelements sehr wichtig, da über ihn der Variablenname im PHP-Skript festgelegt wird. Wird das Element beispielsweise *var* genannt, dann heißt die Variable im PHP-Skript, die das Formular empfängt, auch `$var`.

```
<FORM ACTION="request.php" METHOD="post">
Name:<BR>
<INPUT TYPE="text" NAME="name">
<BR>
User:<BR>
<INPUT TYPE="text" NAME="user" VALUE="root">
<BR>
Passwort:<BR>
<INPUT TYPE="password" NAME="pass">
<BR>
Ist das ein tolles Formular?<br>
<INPUT TYPE="radio" NAME="formular" CHECKED VALUE="ja">ja
<BR>
<INPUT TYPE="radio" NAME="formular" VALUE="nein">nein
<BR>
Markieren, wenn Sie PHP lernen wollen:
<INPUT TYPE="checkbox" NAME="php" CHECKED>
<BR>
<input TYPE="reset" VALUE="Alles zurücksetzen!">
<input TYPE="submit" VALUE="abschicken!">
</FORM>
```

Listing 7.2: HTML-Formular

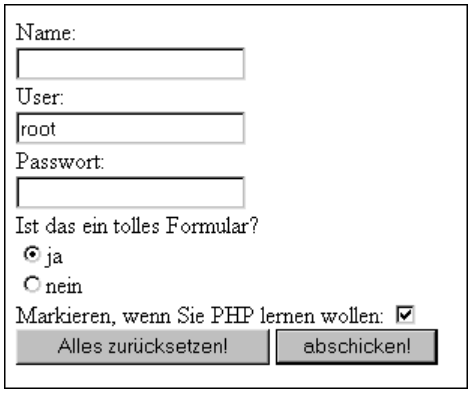


Abbildung 7.1: Beispiel für ein HTML-Formular

Die erste Zeile ist ein einfaches Eingabefeld, das beliebige Einträge aufnehmen kann. Dasselbe gilt auch für das zweite Feld, allerdings ist hier schon ein Voreintrag über das Attribut VALUE vorgenommen worden. Das dritte Eingabefeld gleicht auf den ersten Blick den anderen beiden Feldern. Allerdings ist jeder Eintrag in das Feld verdeckt und der User bekommt nur Sternchen zu sehen. Dies soll verhindern, dass ein eventueller Zuschauer Passwörter ausspähen kann.

Die nächsten beiden Elemente sind zwei Radiobuttons, die miteinander kommunizieren. Wenn der eine Punkt aktiviert ist, wird automatisch der andere deaktiviert. Damit ist es möglich, sich ausschließende Abfragen zu realisieren. Wichtig ist, dass alle Elemente, die zur Gruppe der kommunizierenden Radiobuttons gehören sollen, denselben Wert im Attribut NAME haben.

Die nächste Zeile ist ein Beispiel für eine Checkbox. Sie erlaubt es dem User, Angaben über das Setzen von einfachen Häkchen zu machen. Das das Attribut CHECKED gesetzt wurde, ist die Checkbox bereits markiert.

Die letzte Zeile zeigt zwei Buttons, die zur globalen Steuerung des Formulars verwendet werden. Der ersten Button erlaubt es, das Formular zu löschen und wieder mit den Standardwerten zu versehen. Er wurde als Typ *reset* definiert. Der zweite Button ist ein *submit*-Button und ermöglicht es, die eingetragenen Daten im Formular abzuschicken. Die Zieladresse wurde im FORM-Tag über das Attribut *action* definiert.

Allerdings existiert dieses Skript noch nicht, darum wird der Browser eine Fehlermeldung auswerfen, sobald das Formular abgeschickt wird.

7.3.2 Weitere Formularelemente

Neben dem vielfältigen INPUT-Tag kennt HTML noch zwei weitere Formularelemente, die es erlauben, mit dem User zu interagieren. Zuerst die so genannten Auswahllisten, anhand derer man aus einem Drop-Down-Menü Einträge auswählen kann. Das zweite neue Element ist ein vergrößertes Textfeld, das es ermöglicht, größere Eingaben von Usern entgegenzunehmen. Dieses Eingabefeld dürfte vielen aus Foren oder Mailsystemen bekannt sein.

Tag	Beschreibung
TEXTAREA	erschafft ein mehrzeiliges Texteingabefeld
SELECT	erschafft eine Auswahlliste
OPTION	legt einen Punkt in einer Auswahlliste fest

Tabelle 7.4: Weitere Formularelemente

Genau wie der INPUT-Tag unterstützen diese Formularelemente einige Attribute, die ihr Verhalten im Browser steuern. Im Textfeld können beispielsweise Höhe und Breite festgelegt werden, und in der Auswahlliste haben Sie die Möglichkeit über den Auswahlbereich zu bestimmen.

Attribut	Beschreibung
ROWS	legt die Höhe des Textfeldes in Reihen fest
COLS	legt die Breite des Textfeldes in Zeichen fest
SIZE	legt den Auswahlbereich der Auswahlliste fest
MULTIPLE	legt fest, dass mehrere Punkte in der Auswahlliste über STRG ausgewählt werden dürfen
NAME	legt den Namen des Steuerelements fest

Tabelle 7.5: Attribute zu den neuen Formularelementen

Der OPTION-Tag definiert die Punkte innerhalb einer Auswahlliste. Er hat nur innerhalb des Bereiches eines SELECT-Tag Gültigkeit. Für jeden

Auswahlpunkt muss ein eigener OPTION-Tag angelegt werden, der über das Attribut VALUE den Übergabewert an das Skript bestimmt. Die Ausgabe in der Liste wird über den OPTION-Tag angegeben. Wird kein VALUE-Attribut gesetzt, dann übergibt das Formular automatisch den Ausgabewert in der Liste.

```
<FORM ACTION="request.php" METHOD="post">
Wählen Sie die Art der Anfrage:<BR>
<SELECT NAME="anfrage" SIZE=3 MULTIPLE>
<OPTION VALUE="1">Frage</OPTION>
<OPTION VALUE="2">Lob</OPTION>
<OPTION VALUE="3">Kritik</OPTION>
<OPTION VALUE="4">Beschwerde</OPTION>
<OPTION VALUE="5">Wunsch</OPTION>
</SELECT>
<BR><BR>
<TEXTAREA NAME="text" ROWS="5" COLS="20">Hier ihren Text einge-
ben...</TEXTAREA>
<BR>
<INPUT TYPE="submit">
</FORM>
```

Listing 7.3: Weitere Formular-Elemente

Abbildung 7.2: Weitere Formular-Elemente

Im HTML-Code wurde die Höhe für die Auswahlliste auf drei festgelegt. Da es sich aber um fünf Punkte in der Liste handelt, werden vom Browser automatisch Scrollbalken angefügt, die es erlauben, in der Liste zu navigieren.

Wenn Sie mehr als einen Punkt in der Liste auswählen wollen, dann ist das über die Steuerungstaste (STRG) möglich. Wenn Sie diese gedrückt halten und weitere Punkte auswählen, dann bleiben die anderen Punkte markiert.

Das zweite Eingabefenster ist eine Textbox, in der schon ein vordefinierter Text eingetragen ist. Im Gegensatz zum INPUT-Tag muss der Text allerdings zwischen den öffnenden und schließenden TEXTAREA-Tags stehen, damit er erkannt wird. Auch hier erweist sich der Browser als sehr komfortabel, da er automatisch Scrollbalken schafft, wenn der Text über die unterste Zeile hinausgeht.

7.3.3 Technischer Ablauf einer Formularanfrage

Die Verarbeitung eines Formulars geschieht in mehreren Schritten. Zuerst muss selbstverständlich der Server die HTML-Seite mit dem Formular an den Browser schicken, damit der User die Möglichkeit hat, die vorgegebenen Felder auszufüllen.

Ist das geschehen, klickt der User auf den Button, der die eingetippten Daten wieder an den Server zurückschickt. Das Ziel wird über das Attribut ACTION im FORM-Tag festgelegt. Die Art der Datenübertragung wird über das Attribut METHOD bestimmt.

Der Server empfängt die HTTP-Anfrage und leitet diese an die Serversoftware weiter, die daraufhin das angegebene PHP-Skript startet. Die übergebenen Daten werden als Umgebungsvariablen gespeichert, so dass der PHP-Interpreter sie auslesen kann.

Dem aufgerufenen Skript stehen die übergebenen Werte nun als Variablen zur Verfügung und können verarbeitet werden. Mit diesen Informationen kann eine dynamische Seite erstellt werden, die wieder an den HTTP-Client zurückgeschickt wird.

Sobald das Skript auf dem Server abgearbeitet wurde, werden die übergebenen Daten wieder gelöscht, um den Speicher nicht unnötig zu belasten. Um die Daten also länger speichern zu können, muss man andere Wege gehen. Eine Möglichkeit wäre zum Beispiel eine Datenbank.

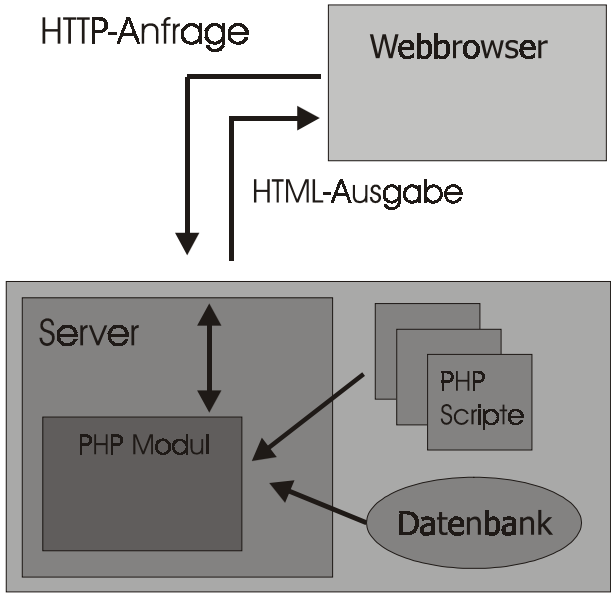


Abbildung 7.3: Verarbeitung eines Formulars

7.3.4 Datenübertragung mit POST und GET

Die Art der Übertragung spielt eine wichtige Rolle in der Webprogrammierung, auch wenn beide Wege von PHP im Prinzip gleich behandelt werden. Die Daten können entweder über die GET- oder über die POST-Methode verschickt werden. Beide Methoden haben ihre Vor- und Nachteile.

Die GET-Methode

Wird die GET-Methode verwendet, dann muss der Kopf des HTML-Formulars diese Form haben:

```
<FORM ACTION="script.php" METHOD="GET">
```

Die Anführungszeichen können auch weggelassen werden, allerdings kann es sein, dass dies nicht von allen Browsern akzeptiert wird.

Die Formulardaten werden bei der GET-Methode als Querystring an die Ziel-URL im Browser angehängt. Die eigentliche Adresse wird von den Daten durch ein Fragezeichen getrennt. Der Übergabestring hat dabei den Aufbau einer Schlüssel-Wert-Paarbildung, die durch das &-Zeichen getrennt sind.

```
Schlüssel1=Wert1&Schlüssel2=Wert2&Schlüssel3=Wert3& ...
```

Jeder Schlüssel entspricht dabei dem Namen eines Formularelements, während der dazugehörige Wert den eingetragenen Daten entspricht. Eventuelle Sonder- oder Leerzeichen werden durch die HTP-Codierung maskiert, die den Netzwerktransport regelt. Diese Codierung wird automatisch durch den PHP-Interpreter wieder aufgeschlüsselt und braucht uns bis auf weiteres nicht zu kümmern.

Ein einfaches Beispiel für die URL eines HTTP-GET-Requests könnte so aussehen:

```
http://www.kulturbrand.de/
script.php?name=Dirk&user=root&pass=geheim&service=login
```

Die Variablen `name`, `user`, `pass` und `service` werden in dieser URL mit den entsprechenden Werten an das Programm script.php übergeben.

Der Nachteil dieser Technik wird schnell deutlich, wenn man bedenkt, dass einige Browser die Länge der URL im Browser begrenzen und den Rest einfach abschneiden. So kann es passieren, dass vor allem bei langen Datensätzen Teile der Informationen verloren gehen.

Ein weiterer Punkt ist die direkte Einsicht in den URL-String mit den übergebenen Daten. Passwörter, die im Formular noch unsichtbar eingegeben wurden, sind im String als Klartext zu lesen. Die GET-Methode eignet sich also nur für wenige Informationen, die keinerlei Sicherheitsrelevanz haben.

Der Vorteil der GET-Methode ist die Möglichkeit, Daten ohne Formular an eine Seite weiterzugeben. Die entsprechenden Schlüssel-Wert-Paare müssen einfach an die URL angehängt werden und können dann auf der Zielseite wieder ausgelesen werden.

Die POST-Methode

Um die POST-Methode zu verwenden, muss dies im FORM-Tag als Methode angegeben werden.

```
<FORM ACTION="script.php" METHOD="POST">
```

Im Gegensatz zur GET-Methode werden die Formulardaten nicht über URL transportiert, sondern einfach an die Standardeingabe des Servers geliefert. Die Daten werden also wie normale Usereingaben behandelt

und können als Parameter in ein Skript übernommen werden. Für ein PHP-Skript macht es keinen Unterschied, ob die Daten per POST oder GET geliefert wurden, da sie vom Interpreter automatisch aufbereitet werden.

Der Unterschied zu GET besteht darin, dass der User die übergebenen Daten nicht zu Gesicht bekommt. Der Länge des Übergabestrings ist ebenfalls keine Grenze gesetzt, so dass die POST-Methode sich auch für große Datenmengen eignet.

Der Nachteil von POST liegt in der Handhabung, weil ein Formular praktisch die einzige Möglichkeit ist einen HTTP-POST-Request zu initiieren. Im Zweifels benutzt man allerdings immer besser die POST-Methode, weil sie schlicht eleganter in der Anwendung ist.

7.3.5 Was wird übergeben?

Je nach Formularelement übergibt das Formular bestimmte Daten an den Webserver. Bei einem einfachen Eingabefeld besteht der Übergabestring einfach aus den Daten, die vom User eingegeben wurden.

Bei komplexeren Formularelementen sieht die Sache allerdings anders aus. Eine Checkbox beispielsweise übergibt nicht die Aussage, für die sie steht, sondern lediglich eine Statusmeldung. Dasselbe gilt für Radiobuttons, die nur den Zustand ein oder aus kennen.

Formular-element	Übergabewert	Beschreibung
`text`	Eingabestring	Ein Textfeld übergibt den Wert, der vom User eingetragen wurde.
`password`	Eingabestring	Ein Passwortfeld übergibt den Wert, der vom User eingetragen wurde.
`hidden`	value-String	Ein verstecktes Feld übergibt den Wert, der als value angegeben wurde.
`checkbox`	on oder value-String	Ist die Checkbox aktiviert, wird »on« bzw. der value-String übergeben.

Tabelle 7.6: Übergabewerte der Formularelemente

Formular-element	Übergabewert	Beschreibung
`radio`	value-String	Es wird der value-String des aktivierten Radiobuttons übergeben.
`select`	value-String	Es wird der value-String des gewählten Listeneintrags übergeben. Wurden mehrere Punkte ausgewählt, wird für jeden Punkt ein eigener String übergeben.

Tabelle 7.6: Übergabewerte der Formularelemente

Es ist wichtig zu wissen, welches Element was übergibt, damit man die Übergabestrings des Formulars korrekt behandeln kann. Wenn man das Formular mit der GET-Methode übergibt, kann man die Übergabestrings leicht überprüfen.

7.3.6 Zugriff auf Formulardaten mit PHP

Wie die letzten Kapitel gezeigt haben, werden die Daten eines Formulars entweder per GET oder per POST an den Server gesandt. Das PHP-Modul startet automatisch das dazugehörige Skript und importiert die Daten als Variablen. Der Variablenname wird durch das Attribut NAME festgelegt und vom Interpreter übernommen.

Es ist wichtig, darauf zu achten, dass nur gültige PHP-Bezeichnungen aus einem Formular als Variable übergeben wird, da die Daten sonst verloren gehen. Ein HTML-Formular akzeptiert zwar Namen wie »Größe« oder »1.Platz«, aber diese werden nicht korrekt in PHP übernommen.

Auswertung von INPUT-Feldern

Im folgenden Beispiel wird ein einfaches Formular ausgewertet, das nur aus INPUT-Feldern und einem Button zum Senden besteht. Das Listing besteht aus zwei Teilen, wobei der erste Teil reines HTML ist, während der zweite Teil durch PHP realisiert wird.

Diese Aufteilung ermöglicht es uns, das Formular in derselben Datei auszuwerten und das Skript zusammenzufassen. Zu diesem Zweck wird das Attribut ACTION auf den eigenen Dateinamen gesetzt. Als Methode wird POST angegeben, um die eingegebenen Daten unsichtbar zu senden.

Die Zusammenfassung von Formular und Skript ist keine Notwendigkeit, da die Daten auch von jedem beliebigen anderen Skript ausgewertet werden könnten. Allerdings bietet sich dieses Vorgehen an, um ein unnötiges Dateiwirrwarr zu vermeiden.

Der HTML Code hat die folgende Form:

```
<FORM ACTION="listing74.php" METHOD="post">
Name:<BR>
<INPUT TYPE="text" NAME="name">
<BR>
User:<BR>
<INPUT TYPE="text" NAME="user" VALUE="root">
<BR>
Passwort:<BR>
<INPUT TYPE="password" NAME="pass">
<BR><INPUT TYPE="submit" VALUE="abschicken">
</FORM>
```

Listing 7.4: HTML-Teil

Die Auswertung des Formulars übernimmt das folgende Listing, das in derselben Datei einfach darunter steht. Da das Formular auf sich selbst verweist, werden die Daten an das PHP-Skript übergeben.

```
<?
echo "Name: $name<br>";
echo "Username: $user<br>";
echo "Passwort: $pass";
?>
```

Listing 7.5: PHP-Teil

Ein denkbar einfaches Skript, das die übergebenen Variablen ausgibt. Die Variablennamen definieren sich einfach über die Namen der einzelnen Formularelemente, die über das Attribut NAME festgelegt wurden. Die erste Seite dürfte etwa so aussehen wie in Abbildung 7.4.

Je nach Konfiguration des PHP-Interpreters können auch einige Warnmeldungen ausgegeben werden, die darauf hinweisen, dass die Variablen nicht definiert sind. Das ändert sich allerdings sehr schnell, wenn man das Formular ausfüllt und abschickt (Abbildung 7.5).

Abbildung 7.4: Formular vor dem Absenden

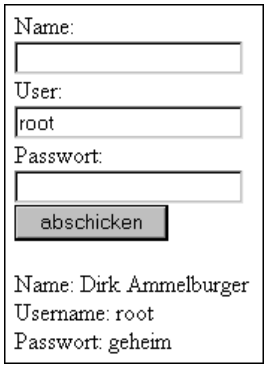


Abbildung 7.5: Formular nach dem Absenden

Die eingetragenen Daten wurden über die Variablen `$name`, `$user` und `$pass` in das Skript importiert und dann wieder ausgegeben. Der Umgang mit Formularen ist also so einfach wie der Umgang mit Variablen selbst, da der Programmierer sich weder um den Transport noch über die korrekte Übergabe der Daten kümmern muss.

Die Daten werden einfach unter dem Namen, der im Attribut NAME festgelegt wurde, an das Skript übergeben und können dann bequem genutzt werden.

Auswertung von Checkboxen und Auswahllisten

In einem der letzten Kapitel wurde geklärt, wie Checkboxen eingesetzt werden und welche Möglichkeiten man damit hat. Wird dieses Formular-

element ohne eine Angabe für das Attribut VALUE gesetzt, dann übergibt eine Checkbox einfach den String »on«, falls sie aktiviert wurde.

Um andere Werte zu übergeben, muss für jede Checkbox das Attribut VALUE auf den entsprechenden Wert gesetzt werden. Ein kurzes Beispiel demonstriert dies:

```
Folgende Namen gehören zu meinen Lieblingsnamen:
<FORM ACTION="listing75.php" METHOD="post">
<INPUT TYPE="checkbox" NAME="box[]"
ALUE="Dirk">Dirk<BR>
<INPUT TYPE="checkbox" NAME="box[]"
ALUE="Niko">Niko<BR>
<INPUT TYPE="checkbox" NAME="box[]"
VALUE="Gustav">Gustav<BR>
<INPUT TYPE="checkbox" NAME="box[]" VALUE="Hans-Egon">Hans-Egon<BR>
<INPUT TYPE="checkbox" NAME="box[]" VALUE="Julia">Julia<BR>
<INPUT TYPE="checkbox" NAME="box[]" VALUE="Gaby">Gaby<BR>
<INPUT TYPE="checkbox" NAME="box[]" VALUE="Lise">Lise<BR>
<INPUT TYPE="checkbox" NAME="box[]" VALUE="June">June<BR>
<INPUT TYPE="submit" VALUE="go">
</FORM><BR>
<?
foreach ($box as $var)
{
        echo $var ." ";
}
?>
```

Listing 7.6: Auswertung von Checkboxen

Ein weiterer interessanter Aspekt ist der Einsatz von Arrays bei der Übergabe von Datenmengen, die zusammengefasst werden können. PHP erkennt eine Übergabe als Array, wenn der Name durch eckige Klammern gekennzeichnet ist. Ist dies der Fall, dann werden alle Daten, die denselben Namen haben, in einem Array zusammengefasst.

Der Einsatz von Arrays bietet sich besonders dann an, wenn multiple Auswahllisten ausgewertet werden sollen. Normalerweise würde eine Mehrfachauswahl die Übergabedaten für das PHP-Skript teilweise überschreiben, da sie alle mit demselben Namen übergeben werden.

Diese Problematik kann über ein Array elegant umgangen werden.

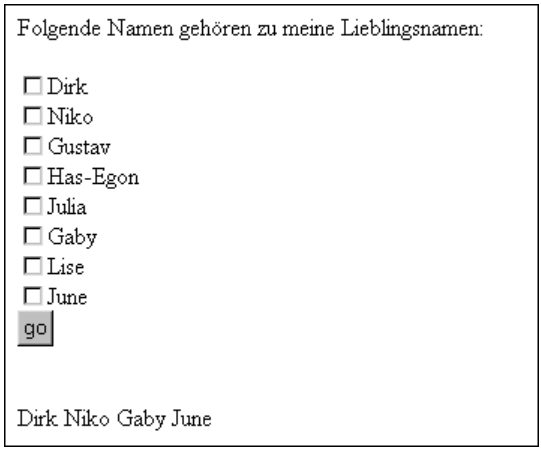


Abbildung 7.6: Ausgabe von Listing 7.6

```
Die folgenden Namen gehören zu meinen Lieblingsnamen:<BR>
<FORM ACTION="listing76.php"  METHOD="post">
<SELECT NAME="liste[]" MULTIPLE>
<OPTION VALUE="Dirk">Dirk</OPTION>
<OPTION VALUE="Niko">Niko</OPTION>
<OPTION VALUE="Gustav">Gustav</OPTION>
<OPTION VALUE="Hans-Egon">Hans-Egon</OPTION>
<OPTION VALUE="Julia">Julia</OPTION>
<OPTION VALUE="Gaby">Gaby</OPTION>
<OPTION VALUE="Lise">Lise</OPTION>
<OPTION VALUE="June">June</OPTION>
</SELECT>
<INPUT TYPE="submit" VALUE="go">
</FORM>

<?
foreach ($liste as $var)
{
        echo $var ." ";
}
?>
```

Listing 7.7: Auswertung einer Auswahlliste

Das Formular übergibt ein Array mit dem Namen `liste[]`, das alle ausgewählten Punkte der Auswahlliste enthält. Über die `foreach()`-Schleife werden die einzelnen Elemente der Liste ausgegeben und durch Leerzeichen getrennt.

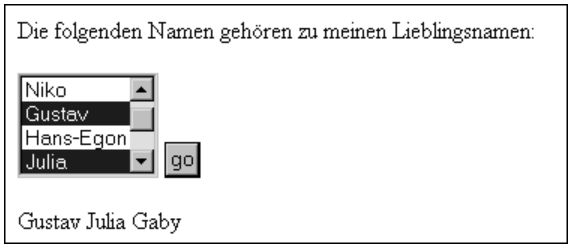


Abbildung 7.7: Ausgabe von Listing 7.7

7.3.7 Formulardaten über globale Arrays

Die Konfigurationsdatei für den PHP-Interpreter php.ini gibt dem Administrator die Möglichkeit die automatische Datenübergabe als Variable an ein Skript zu unterbinden. Sollte das der Fall sein, können Formulardaten trotz allem ausgelesen werden, da PHP automatisch assoziative Arrays mit allen Übergabewerten anlegt.

Es wird zwischen den GET-Werten und den POST-Werten im Skript unterschieden, da der Interpreter die Daten unterschiedlich behandeln muss. Darum gibt es für jede Datengruppe ein eigenes Array.

Die GET-Variablen werden automatisch im globalen Array $HTTP_GET_VARS abgelegt und die POST-Variablen speichert der Interpreter im Array $HTTP_POST_VARS. Beide Arrays sind assoziativ geordnet, so dass man auf jeden Wert über seinen Namen zugreifen kann.

Das folgende Beispiel zeigt den Einsatz dieser Arrays. Das Formular ist aus einem der vorhergehenden Kapitel entnommen.

```
<FORM ACTION="listing77.php" METHOD="post">
Name:<BR>
<INPUT TYPE="text" NAME="name">
<BR>
User:<BR>
<INPUT TYPE="text" NAME="user" VALUE="root">
<BR>
Passwort:<BR>
<INPUT TYPE="password" NAME="pass">
<BR><INPUT TYPE="submit" VALUE="abschicken"><BR>
</FORM>

<?
```

```
echo $HTTP_POST_VARS['name'];
echo "<BR>";
echo $HTTP_POST_VARS['user'];
echo "<BR>";
echo $HTTP_POST_VARS['pass'];
?>
```

Listing 7.8: Formulardaten aus globalen Arrays

Das Programm bewirkt genau dasselbe wie das erste Beispielprogramm mit Formularen. Der Unterschied besteht im Datenhandling, das nun über die Arrays gehandhabt wird.

Ein weiteres Einsatzszenario für diese Technik ist die bessere Kontrolle über alle Variablen, die dem Skript übergeben wurden. So kann zum Beispiel gleich zu Anfang des Skripts festgestellt werden, welche und vor allem wie viele Werte in das Programm importiert werden. Diese Information kann bei sicherheitsrelevanten Programmen sehr wichtig sein.

Ein weiterer Punkt ist die Vermeidung von Namenskonflikten im Programm. Wenn Übergabevariablen nur über das Array behandelt werden, ist die Möglichkeit von Datenverlusten nicht gegeben. Außerdem bleiben alle fremden Daten »handlich« und können einfach verwaltet werden.

Die simpelste Kontrolle der übergebenen Daten ist die Ausgabe aller Variablen. Ohne das Array wäre diese Aufgabe weit schwieriger zu lösen.

```
<?
if (sizeof($HTTP_GET_VARS) != 4) {die("Falsche
Argumentanzahl!");}
foreach ($HTTP_GET_VARS as $key => $element)
{
        echo "$key: $element<br>";
}
?>
```

Listing 7.9: Kontrolle der Übergabeparameter

Mögliche Ausgabe:

```
name: Dirk
nachname: Ammelburger
ort: München
beruf: Autor
```

Die Kontrolle der Anzahl der Elemente kann ganz einfach über die Funktion `sizeof()` realisiert werden. Wird eine bestimmte Anzahl von Elementen unter- oder überschritten, dann gibt das Programm eine Fehlermeldung aus.

7.3.8 Datenübergabe ohne Formular

Die letzten Kapitel haben gezeigt, dass die GET-Methode Formulardaten über die URL weitergibt. Diese Tatsache erlaubt es, Daten an ein PHP-Programm zu übergeben, ohne ein Formular zu verwenden. Ein einfacher Link genügt, der in der URL die entsprechenden Schlüssel-Wert-Paare weitergibt.

Ein Formular erzeugt die entsprechende URL für die Datenübergabe automatisch, ohne dass Sie sich darum kümmern müssen. Wollen Sie Daten ohne ein Formular versenden, müssen Sie den Aufbau der URL kopieren und den String von Hand erstellen. Eine typische GET-URL könnte so aussehen.

```
http://www.kulturbrand.de/php/link.php?linkid=1
```

Wird dieser Link über die `<A>`-Tag in eine HTML-Seite eingebaut und angeklickt, dann übergibt er automatisch die Variable `$linkid` mit dem Wert 1 an das Programm link.php.

Der HTML-Tag mit dem man Hyperlinks setzt ist relativ einfach aufgebaut. Er besitzt im Wesentlichen zwei Attribute: auf der einen Seite die Zieladresse der verlinkten Webpage und auf der anderen Seite eine Angabe, in welchem Browserfenster das Ziel angezeigt werden soll.

Attribut	Angaben	Erklärung
`HREF`	Ziel URL	Die Ziel URL ist die Webadresse, zu der der Link führt. Die Angabe kann relativ zur aktuellen Position erfolgen oder als komplette Adresse angeben werden.
`TARGET`	Zielseite im Browser	Es wird entweder die Bezeichnung eines Browserfensters bzw. Frames angegeben oder `_blank`, falls ein neues Fenster geöffnet werden soll.

Tabelle 7.7: Attribute für Hyperlinks

Die klickbare Ausgabe im Browser wird zwischen den öffnenden und schließenden Link-Tag gestellt. Ein typischer Link könnte in HTML so aussehen:

```
<A HREF="link.php" TARGET="_blank">Klick mich!</a>
```

Ausgabe:

Klick mich!

Wird die Zieladresse um einige Schlüssel-Wert-Paare erweitert, werden diese automatisch mitübergeben. Ein PHP-Skript wertet sie genauso wie eine Formularausgabe. Es wird Zeit für ein kleines Beispiel:

```
<A HREF="listing79.php?link=1">Link 1</A><BR>
<A HREF="listing79.php?link=2">Link 2</A><BR>
<A HREF="listing79.php?link=3">Link 3</A><BR>
<A HREF="listing79.php?link=4">Link 4</A><BR>
<A HREF="listing79.php?link=5">Link 5</A><BR><BR>
<?
switch($link)
{
        case 1: echo "Sie haben den ersten Link angeklickt!"; break;
        case 2: echo "Sie haben den zweiten Link angeklickt!"; break;
        case 3: echo "Sie haben den dritten Link angeklickt!"; break;
        case 4: echo "Sie haben den vierten Link angeklickt!"; break;
        case 5: echo "Sie haben den fünften Link angeklickt!"; break;
}
?>
```

Listing 7.10: Datenübergabe ohne Formular

Je nachdem, welchen Link Sie anklicken, wird ein anderer Wert in der Variable `$link` übergeben. Eine `switch()-case`-Konstruktion überprüft den Wert von `$link` und gibt eine entsprechende Statusmeldung aus.

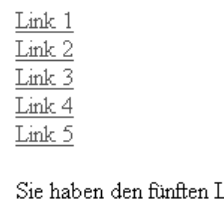


Abbildung 7.8: Datenübergabe ohne Formular

Natürlich ist das Programm so nicht sehr sinnvoll, aber die Möglichkeiten werden auf jeden Fall klar. Anstelle einer Statusmeldung könnte eine komplett andere Seite aufgebaut werden, die dem User erlaubt weiterzunavigieren.

Oft ist es sehr viel eleganter und auch einfacher, ein Skript über einen Link aufzurufen, anstatt jedes Mal ein Formular zur programmieren. Formulare sind nur dann sinnvoll, wenn wirklich Usereingaben abgefragt werden müssen, ansonsten können Links verwendet werden.

7.3.9 Formulare prüfen

Die Geschichte vom DAU dürfte allgemein bekannt sein, denn er ist inzwischen eine feste Konstante in der Softwareentwicklung geworden. Ein Programm ist erst dann wirklich gut, wenn der **D**ümmste **A**nzunehmende **U**ser in der Lage ist damit umzugehen, ohne Schaden anzurichten.

Aus diesem Grund ist es wichtig, ein Programm auf alle möglichen Fehler zu überprüfen und auch ausgefallene Userreaktionen zu berücksichtigen. Die größte Fehlerquelle ist die Interaktion mit dem User, die im Web in der Regel über Formulare geschieht. Ein Programm sollte immer in der Lage sein fehlerhafte Formulareinträge herauszufiltern und Fehlermeldungen auszugeben. Die Prüfung von Formularen ist in der Praxis elementar.

Ich bin im Internet einmal über ein Formular gestolpert, das Linkeinträge aufgenommen hat. Der User wurde aufgefordert seinen Lieblingslink anzugeben und abzuschicken. Das Problem dieses Programms war ein Denkfehler des Autors, der davon ausgegangen ist, dass jeder User seinen Link mit http:// am Anfang einträgt. Dem war nicht so, und ich gebe auch gerne zu, dass ich ebenfalls zur Gruppe der DAUs gehört habe, die diesen Teil vergessen haben.

Das Ergebnis dieser Geschichte war eine dynamisch erzeugt Linkliste, in der kein Link funktioniert hat, weil das Programm die Eingaben nicht überprüft hatte.

Damit Ihnen solche Erfahrungen erspart bleiben, sollte jede Usereingabe kontrolliert werden. Gerade Strings wie eingetragene Links oder E-Mail-Adressen können recht einfach mit den PHP-Stringfunktionen kontrolliert werden.

Funktion	Beschreibung
strpos()	Findet die Position einer Zeichenkette in einem String
strrpos()	Findet das letzte Auftreten eines einzelnen Zeichens in einem String
strstr()	Findet das erste Auftreten einer Zeichenkette und gibt diese mit allen folgenden Zeichen zurück
substr()	Gibt einen Teilbereich eines Strings zurück
strcmp()	Vergleicht zwei Strings
strcasecmp()	Vergleicht zwei Strings unabhängig von der Groß- und Kleinschreibung
strnatcmp()	Vergleicht zwei Strings unter Berücksichtigung von Zahlen
strnatcasecmp()	Vergleicht zwei Strings unter Berücksichtigung von Zahlen, unabhängig von Groß- und Kleinschreibung

Tabelle 7.8: Stringfunktionen zur Kontrolle von Formularen

Ein kleines Beispiel verdeutlicht die Möglichkeiten, die diese Funktionen bieten. Der HTML-Teil dürfte inzwischen bekannt sein:

```
<FORM ACTION="listing80.php" METHOD="post">
Link eintragen:<BR>
<INPUT TYPE="text" NAME="link">
<BR>
eMail-Adresse eintragen:<BR>
<INPUT TYPE="text" NAME="email">
<BR><INPUT TYPE="submit" VALUE="abschicken"><BR>
</FORM>
```

Das Skript besteht im Großen und Ganzen eigentlich nur aus einer Reihe von Bedingungen, die die übergebenen Daten prüfen.

```
<?
//Leereinträge werden abgefangen!
if ($link == "") {die("Link fehlt!");}
if ($email == "") {die("eMail fehlt!");}
//Link wird auf http:// überprüft!
if (strpos(strtolower($link), "http://") != 0) {die ("http://
fehlt!");}
//eMail-Adresse wird überprüft! Ist @ und . enthalten?
if(!(strstr($email, "@"))) {die ("eMail nicht korrekt!");}
if(!(strstr($email, "."))) {die ("eMail nicht korrekt!");}
//Reihenfolge von . und @ wird überprüft!
```

```
if(strpos($email, ".") < strpos($email, "@")) {die ("eMail nicht
korrekt!");}
//Nur .de- und .com-Endungen werden zugelassen!
$i = strrpos($email, ".");      //Position vom Punkt im String!
$str = substr($email, $i, 4);   //Endung wird abgeschnitten!
switch ($str)                   //Kontrolle der Endung!
{
case ".de": break;
case ".com": break;
default: die("eMail nicht zulässig!");
}
echo "Alles Okay!";
?>
```

Listing 7.11: Kontrolle von Formulareinträgen

Die Kommentare sollten das Programm soweit erklären. Die Funktion `strtolower()` wandelt einen String komplett in Kleinbuchstaben um, damit die Kontrolle auf den Stringteil »*http://*« in jedem Fall korrekt funktioniert. Alle anderen Funktionen sind in der oberen Tabelle erläutert.

Natürlich ist dieses Skript nicht erschöpfend in der Kontrolle, aber es deckt bereits die wichtigsten Punkte und Fehlerquellen ab. In einem späteren Kapitel werden die Möglichkeiten der regulären Ausdrücke beschrieben, die sehr viel mächtiger sind als die einfachen Stringfunktionen von PHP. Mit regulären Ausdrücken kann die Eingabekontrolle viel besser umgesetzt werden.

7.3.10 Dynamische Formulare

Der Sinn von PHP liegt in der dynamischen Erschaffung von HTML-Code in Abhängigkeit von Usereingaben. Dieser Vorgang muss sich allerdings nicht nur auf reine Ausgabeelemente wie Tabellen oder Formatierungsangaben beschränken, sondern kann auch neue Möglichkeiten der Interaktion schaffen.

Auf der einen Seite ist es oft leichter, ähnlich aufgebaute Formulare dynamisch zu erzeugen, um Tipparbeit zu sparen. Auf der anderen Seite kann es vorkommen, dass eingegebene Daten die weiteren Abfragen bestimmen und das Formular entsprechend verändert werden muss. Beide Situationen lassen sich mit PHP recht einfach bereinigen.

Ein kleines Beispiel zeigt den Einsatz von dynamischen Formularen:

```
<?
$liste = array("C/C++", "Java", "PHP", "Basic", "ASP", "Pascal");
echo "<form action='listing81.php' method='post'>";
echo "<select name='lang'>";
foreach ($liste as $var)
{
   echo "<option>$var</option>";
}
echo "<input type='submit' value='go'></form><br>";
if (isset($lang))
{
   echo "Wie würden Sie Ihre Kenntnisse in $lang beurteilen (Schulno-
ten)?<br>";
   $liste = range(1,6);
   echo "<form action='listing81.php'method='post'>";
   foreach ($liste as $var)
   {
      echo "<input type='radio' name='note' value='$var'>$var<br>";
   }
   echo "<input type='hidden' name='lang' value='$lang'>";
   echo "<input type='submit' value='abschicken'></form><br>";
}
if (isset($note))
{
   if($lang == "PHP" and $note != 1)
      {echo "Sie halten das richtige Buch in der Hand!";}
   if($lang == "PHP" and $note == 1)
      {echo "Sie halten trotzdem das richtige Buch in der Hand!";}
   if($lang != "PHP" and $note != 1)
      {echo "Bald werden Sie merken, dass PHP besser ist!";}
}
?>
```

Listing 7.12: Ein dynamisches Formular

Das Programm realisiert einen kleinen Fragebogen, der dem User erlaubt Angaben über seine Interessen zu machen. Am Ende erhält er eine mehr oder weniger ernst zu nehmende Beurteilung seiner Situation.

Das Besondere an diesem Skript ist der dynamische Aufbau der Formulare, die direkt aus einem Array heraus formuliert werden. Die Schleife `foreach()` spielt hierbei eine zentrale Rolle. Für jedes Element wird ein neuer Formularpunkt ausgegeben, der vom User gewählt werden kann.

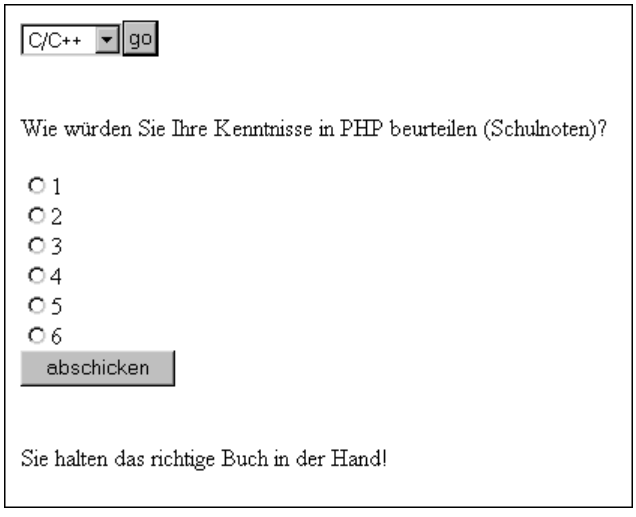


Abbildung 7.9: Fragebogen zu Programmiersprachen

Ein netter Nebeneffekt ist der weitaus geringere Schreibaufwand für das Programm und die einfache Pflege. Es müssen lediglich die Arrays geändert werden, um dem Fragebogen einen neuen Kontext zu geben. In statischen Formularen müsste jede Zeile per Hand angepasst werden.

Ein wichtiger Punkt, auf den Sie unbedingt achten sollten, sind Anführungszeichen in Ausgaben mit `echo`. Da der auszugebende String bereits mit doppelten Anführungszeichen ("...") gekennzeichnet ist, können die HTML-Attribute nicht mehr so abgegrenzt werden. Um Probleme zu umgehen, können die einfachen Anführungszeichen (‚...') verwendet werden, die auf der Tastatur über der #-Taste zu finden sind.

Eine weitere Möglichkeit ist der Einsatz von Backslashs, die mögliche Sonderzeichen maskieren.

```
echo "Er sagte: \"Die Welt ist rund!\".";
```

Ausgabe:

```
Er sagte: "Die Welt ist rund!".
```

Der Interpreter ignoriert die doppelten Anführungszeichen und gibt sie als einfachen Text aus. Ohne die Anführungszeichen hätte es eine Fehlermeldung gegeben.

7.3.11 Übungen

Ü

1. Was ist der Unterschied zwischen GET und POST?
2. Was ist der Unterschied zwischen Checkboxen und Radiobutton?
3. Wie wird das Ziel einer Formularanfrage festgelegt?
4. Welche Variablen übergibt dieses Formular?

```
<FORM ACTION="pwd.php" METHOD="post">
Name:<BR>
<INPUT TYPE="text" NAME="name"><BR>
Passwort:<BR>
<INPUT TYPE="password" NAME="pass">
<BR><INPUT TYPE="submit" VALUE="abschicken">
</FORM>
```

5. Welche Möglichkeiten gibt es noch, um Übergabewerte auszulesen?
6. Welche Schlüssel und welche Werte übergibt dieser Link?

```
<a href="go.php?la=de&site=home&id=7462">klick mich!</a>
```

7. Was ist ein DAU? Was macht ein DAU?
8. Welche Vorteile können dynamische Formulare haben?

7.4 Cookies

PHP erlaubt die Steuerung von Webseiten über Cookies, die es ermöglichen Informationen auf einem Clientrechner zu speichern und zu einem späteren Zeitpunkt wieder auszulesen. Dieser Mechanismus wird über das HTTP-Protokoll gesteuert und gibt Ihrem PC eine Art »Gedächtnis« im Umgang mit dem Internet.

Cookies wurden zu Beginn des Internets, so wie wir es kennen, von der Firma Netscape entwickelt, die so mit der Version 1.1 ihres Browsers einen neuen Standard setzte. Wenn man genau wissen will, warum Netscape den Namen Cookies gewählt hat, dann wird man allerdings enttäuscht. Angeblich war die erschöpfende Aussage der Firma: »For no special reason ...«

Cookies gelten heute noch bei vielen als einer der negativen Aspekte des Netzes, da allerlei merkwürdige Geschichten über diese harmlosen »Kekse« im Umlauf sind. Angeblich würden Firmen damit Userprofile anlegen können und das Surfverhalten eines Internetnutzers ausspionieren.

Ich bin sogar schon mal gefragt worden, wie man sich vor virenverseuchten Cookies schützen kann, ohne dabei ganz aufs Internet verzichten zu müssen.

Diese Schauergeschichten sind natürlich blanker Unsinn und rühren von einer schlechten Informationspolitik der Webseitenbetreiber her. Ein Cookie ist im Prinzip nichts anderes als eine Textdatei, in der vom Browser Informationen gespeichert werden. Bei Interesse können Sie Ihre Cookies mal durchgehen und feststellen, was in ihnen gespeichert wird (siehe Kasten).

Ein Cookie kann immer nur die Informationen enthalten, die von einem Webserver empfangen wurden. Es ist also nicht möglich, dass ein Cookie mehr über Sie verrät, als Sie selbst ins Internet geschickt haben. Die Virengefahr ist genauso ausgeschlossen, da ein Cookie nur Text enthält, der von keinem System ausgeführt wird.

Der einzige Sinn und Zweck dieser Entwicklung ist das Speichern von Daten, um zu einem späteren Zeitpunkt wieder auf diese zurückgreifen zu können. Ein Server »erinnert« sich nicht daran, wann Sie das letzte Mal auf einer Seite waren, und vor allem, was Sie da gemacht haben. Ein prall gefüllter Warenkorb in einem Onlineshop würde demnach einfach verloren gehen, wenn der Server Sie nicht wiedererkennen würde.

Ein Cookie erleichtert also die Identifikation eines Users und kann kurzzeitig Daten zwischenspeichern, um sie später wieder einzusetzen. Diese Charakteristik macht Cookies zu einem praktischen Werkzeug im Aufbau von dynamischen Webseiten.

Wo werden Cookies gespeichert?

Je nach Browser und Betriebssystem werden Cookies an unterschiedlichen Plätzen und in unterschiedlicher Form gespeichert. Ich werde nur auf die beiden wichtigsten Browser (Netscape Navigator und der Microsoft Explorer) eingehen sowie auf den Unterschied zwischen Windows 95/98 und Windows NT/2000.

Microsoft Explorer

Der Microsoft Explorer legt für jeden Cookie eine eigene Textdatei an. Der Name der Datei setzt sich aus dem Benutzernamen des Rechners sowie dem Servernamen, von dem der Cookie gesetzt wurde, zusammen.

Da der Explorer direkt in das Betriebssystem Windows eingebunden ist, werden die Cookies bei den Benutzerdaten abgelegt. Unter Windows 95/98 findet man sie im Windowsverzeichnis in dem Ordner *Cookies*.

Windows 2000- sowie NT-Systeme sind Mehrbenutzerbetriebssysteme und müssen mehrere Userprofile verwalten. Dazu wird im Verzeichnis *Dokumente und Einstellungen* für jeden User ein Ordner angelegt, der seine Daten aufnimmt. Hier werden auch alle Cookies gespeichert, die für diesen User gültig sind.

Netscape Navigator

Der Navigator speichert Cookies nach einem anderen System. Anstatt die Daten in verschiedene Dateien zu packen, wirft Netscape alle Cookies in eine große Datei, die auf den treffenden Namen cookies.txt hört.

Je nach Browserversion oder Betriebssystem liegt diese Datei an einer anderen Stelle. Die alten Netscapeversionen haben die Datei direkt im selben Verzeichnis wie die restlichen Programmdateien gespeichert. Ab der Version 6 werden allerdings auch Mehrbenutzersysteme unterstützt und man findet seine Cookies auch unter *Dokumente und Einstellungen* bei dem jeweiligen Benutzer.

Die Daten liegen allerdings ein wenig versteckt im Verzeichnis Anwendungsdaten unter dem Namen Mozilla, da hier jedes Programm userspezifische Daten ablegen kann.

7.4.1 Was sind Cookies?

Ein paar Worte zur technischen Seite. Cookies werden über den Header einer HTTP-Anfrage gesteuert. Das heißt, ein Browser muss Anweisungen, die Cookies betreffen, vor jeder anderen Ausgabeanweisung erhalten. Es darf also kein `echo`- oder `print`-Befehl vor den Anweisungen für das Setzen oder Lesen eines Cookies stehen.

Vorhandene Cookies werden ähnlich den Formulardaten automatisch in das PHP-Skript importiert. Sie stehen als Variable zur Verfügung und können gleichzeitig aus dem globalen Array `$HTTP_COOKIE_VARS` ausgelesen werden.

Ein Cookie ist eine einfache Textinformation, die maximal eine Größe von 4 Kilobyte haben darf. Das entspricht etwa 4000 ASCII-Zeichen oder

drei bis vier eng beschriebenen Schreibmaschinenseiten. Der Platz reicht definitiv nicht, um größere Bildinformationen oder etwas Ähnliches abzulegen.

Die Lebensdauer von Cookies kann frei bestimmt werden. Sie reicht von einer einmaligen Sitzung des Browsers (Daten werden gelöscht, wenn der Browser geschlossen wird) bis hin zu einer Zeitspanne von mehreren Wochen oder Monaten. Je nach Sinn und Zweck des Cookies kann ein genaues Ablaufdatum bestimmt werden.

Ein Cookie kann nur von dem Server ausgelesen werden, von dem er auch gesetzt wurde. Der Browser gibt die Cookiedaten nur weiter, wenn der Server eindeutig als Quelle der Daten identifiziert wurde. Es ist auch möglich, Cookies nur für einen bestimmten Bereich des Servers freizugeben.

Die Konventionen rund um den Datenverkehr im Internet verbieten, dass mehr als 20 Cookies pro Domain gespeichert werden dürfen. Auch die Anzahl der Cookies insgesamt in einem Browser ist beschränkt. Je nach Version und Modell kann diese Vorgabe allerdings schwanken.

Die Möglichkeit, Cookies zu setzen, ist keine Voraussetzung, auf die man sich immer verlassen kann. Abgesehen von den oben genannten Einschränkungen muss man damit rechnen, dass ein User es nicht zulässt, dass Cookies auf seinem Rechner gespeichert werden. Sie sollten entweder immer eine Alternative bereitstellen (Webpage ohne Cookies) oder den User darauf hinweisen, dass bestimmte Funktionen nur mit eingeschalteten Cookies zugänglich sind.

Mehr Informationen über Cookies finden Sie im Internet auf den Seiten von Netscape unter

`http://www.netscape.com/newsref/std/cookie_spec.html`

7.4.2 Wo werden Cookies eingesetzt?

Die Einsatzmöglichkeiten von Cookies sind recht breit gestreut. Grundsätzlich dient der Einsatz von Cookies dazu, eine Webseite zu personalisieren und speziell an den User anzupassen. Jeder kennt bestimmt das nette Gimmick von Amazon, das jeder Kunde auf der Startseite mit Namen begrüßt wird. Das ist allerdings nur möglich, wenn man schon einmal etwas gekauft oder seine Daten anderweitig angegeben hat.

Da die Daten lokal gespeichert werden, ist der Aufwand für den Programmierer minimal, da kaum Serverressourcen beansprucht werden. Der Browser schickt automatisch bei jeder Sitzung die entsprechenden Daten und das Skript muss lediglich die Werte ausgeben.

Webseiten individuell gestalten

Jeder User geht mit anderen Wünschen und Vorstellungen auf eine Seite. Es ist nie leicht, es allen recht zu machen, wenn die Seite auf einen gemeinsamen Nenner gebracht werden muss. Daher ist es besonders elegant, wenn man Cookies dazu verwendet, die Seite ganz nach den Wünschen des Benutzers einzurichten.

Bemerkt ein Skript zum Beispiel, dass sich der User fast immer nur auf den Sportseiten des Webangebotes bewegt, dann kann beim nächsten Besuch die Startseite gleich alle relevanten Sportdaten präsentieren. Ein anderer Surfer, der beispielsweise nur Börsendaten abruft, wird in Zukunft immer nur Börsendaten auf der Startseite finden.

Auch den geschäftlichen Aspekt sollte man hierbei nicht vergessen, denn was ist dümmer als einem Fußballfan Tennissocken verkaufen zu wollen? Ein Onlineshop kann das Angebot speziell für den Kunden zusammenstellen, indem er einfach alle Produkte, die in das Interessenprofil des Users passen, gleich auf den ersten Seiten präsentiert.

Einkaufen im Netz

Eine weitere, häufig genutzte Möglichkeit von Cookies sind Warenkörbe für Onlineshops. Jedes Produkt, das der Kunde auswählt, wird in einem Cookie gespeichert. So stehen die Daten auf jeder Seite des Servers zur Verfügung, ohne dass das Skript sie umständlich übergeben muss. Selbst wenn der User den Rechner ausschaltet und am nächsten Tag wiederkommt, ist der Warenkorb noch vorhanden.

7.4.3 Aufbau von Cookies

Wie bereits erwähnt wurde, werden Cookies als HTTP-Header an den Client (Browser) gesendet und verarbeitet. Dabei müssen eine Reihe von Werten übergeben werden, die das Verhalten des Cookies steuern. Neben dem Namen und einigen anderen Daten, spielt vor allem die Lebensdauer des Cookies eine wichtige Rolle. Folgende Werte können übergeben werden:

Wert	Beschreibung
`name`	name des Cookies
`expires`	verfallsdatum des Cookies
`path`	legt den Pfad auf dem Server fest, an den der Cookie gesendet werden darf
`domain`	legt die Domain fest, für die der Cookie gültig ist
`secure`	beschränkt den Cookie auf SSL-Verbindungen

Tabelle 7.9: Cookiedaten

Bis auf den Namen des Cookies sind alle Werte optional. Werden sie nicht angegeben, dann wird die Standardeinstellung für Cookies genutzt.

Der **Name** des Cookies legt die Bezeichnung fest, unter der die Daten später als Variable ins Skript übertragen werden. Für die Benennung gelten dieselben Regeln wie für die Bezeichner von Variablen in PHP.

Über **expires** wird festgelegt, wann der Cookie seine Gültigkeit verliert und gelöscht werden soll. Das Datum muss dabei einem ganz bestimmten Format entsprechen und kann nur in der Zeitzone GMT angegeben werden.

```
dd-mon-YYYY hh:mm:ss GMT
```

Möchte man ein Cookie vorzeitig löschen, dann muss man das Datum mit einem Wert überschreiben, der vor dem aktuellen Datum liegt. Wird kein Datum angegeben, dann verfällt der Cookie, sobald der Browser geschlossen wird.

Der Serverpfad wird über **path** festgelegt und setzt fest, wo der Cookie genau gültig ist. Unterverzeichnisse werden automatisch eingeschlossen. Möchte man den gesamten Server für gültig erklären, dann gibt man das Root-Verzeichnis des UNIX-Systems an, nämlich /

Wird kein Pfad angegeben, gilt das aktuelle Verzeichnis.

Die **domain** für ein Cookie legt den Servernamen fest, an die der Browser die Cookiedaten senden darf. In der Regel ist das derselbe Server, der den Cookie auch gesetzt hat. Es ist allerdings auch möglich, ein Cookie für einen anderen Server zu setzen. Der ursprüngliche Server kann die Daten dann nicht mehr ändern.

Wird **secure** mit an den Browser übergeben, dann werden die Cookiedaten nur über eine verschlüsselte Leitung gesandt. Die Übertragung muss per SSL (Secure Sockets Layer) erfolgen, sonst gilt die Leitung als unsicher. Das SSL-Protokoll wurde entwickelt, um die Geheimhaltung zwischen zwei Servern zu gewährleisten. Das Protokoll hat sich inzwischen als Standard im Internet durchgesetzt und ist als Freeware erhältlich.

7.4.4 Einsatz von Cookies in PHP

Der Einsatz von Cookies in PHP ist sehr einfach. Es wird lediglich eine Funktion benötigt, die alle Übergabeparameter für das Setzen eines Cookies zusammenfasst. Die Funktion lautet `setcookie()` und wird mit sechs Parametern aufgerufen:

```
setcookie(name, value, expires, path, domain, secure)
```

Die Parameter dürften aus dem vorhergehenden Kapitel bekannt sein, allerdings hat PHP zwei Punkte vereinfacht. Anstelle eines exakt formatierten Datums muss lediglich ein UNIX-Timestamp an die Funktion übergeben werden. Die entsprechende Formatierung wird von PHP automatisch erstellt.

Der zweite Punkt ist die Angabe einer SSL-Verbindung. Es genügt, der Funktion entweder TRUE oder FALSE zu übergeben, um die Situation zu klären.

Da es sich bei Cookies um einen Bestandteil des HTTP-Headers handelt, darf vor dem Funktionsaufruf keine Ausgabe an den Browser erfolgen. Das Sprachkonstrukt `echo` oder die Funktion `print()` sind also vor der Funktion tabu.

Wurde ein Cookie gesetzt, steht er beim nächsten Aufruf des Skripts automatisch als Variable zur Verfügung. Es bedarf also keiner weiteren Funktion um die Daten wieder auszulesen. PHP behandelt Cookies genau wie die Übergabedaten aus Formularen oder Links.

```
<?
if(isset($myCookie))
{
    echo "\$myCookie hat den Wert: $myCookie";
}
else
{
```

```
    $expires = time() + 3600;
    setcookie("myCookie", "Hallo PHP!", $expires, "",
"", FALSE);
    echo "Cookie wurde gesetzt!";
}
?>
```

Listing 7.13: Ein Cookie wird gesetzt

Grundvoraussetzung für dieses Programm ist natürlich, dass der Browser Cookies akzeptiert. Ist das der Fall, dann wird je nach Konfiguration eine Warnmeldung ausgegeben, die genauere Auskunft über den Cookie gibt.

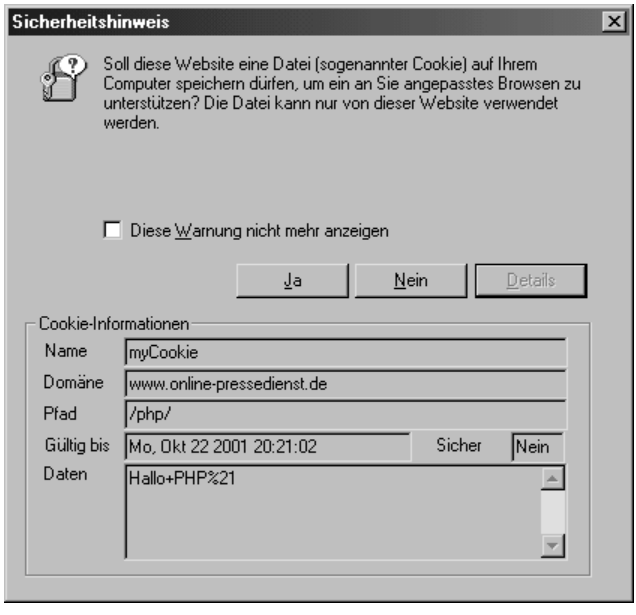


Abbildung 7.10: Sicherheitsabfrage im Microsoft Explorer

Da beim ersten Aufruf des Programms noch kein Cookie vorhanden ist, wurde die Variable `$myCookie` nicht gesetzt. Aufgrund dessen wird beim ersten Start der else-Block der `if()`-Bedingung ausgeführt, der die Anweisungen für den Cookie enthält.

Die erste Anweisung setzt die Variable `$expires` auf den Wert des aktuellen Timestamps, der um 3600 Sekunden (eine Stunde) erhöht wird. Damit wird die Lebensdauer des Cookies festgelegt.

Die Funktion `setcookie()` erstellt daraufhin einen Cookie mit dem Namen *myCookie,* der den String »Hallo PHP!« speichert. Als Verfallsdatum wird die oben berechnete Variable `$expires` angegeben. Für den Pfad und die Domain werden keine Angaben gemacht, da für die Zwecke des Programms die Defaultwerte ausreichen. Der letzte Parameter legt die Sicherheitsstufe fest. Die Übergabe von FALSE bedeutet, dass wir auf eine SSL-Leitung verzichten.

Bei jedem weiteren Aufruf des Skripts wird der Wert des Cookies in der Variable `$myCookie` übergeben. Da die Variable jetzt gültig ist, gibt das Skript einfach ihren Wert aus. Erst nach einer Stunde muss der Cookie neu gesetzt werden.

7.4.5 Das Array $HTTP_COOKIE_VARS

Genau wie Formulardaten speichert PHP alle gültigen Cookies in einem globalen Array. Es ist also möglich, alle übergebenen Cookies zentral zu verwalten und über das Array $HTTP_COOKIE_VARS auszugeben. Das Array ist assoziativ, so dass jeder Wert über den Namen bestimmt werden kann.

```
<?
if(isset($myCookie))
{
    $data = $HTTP_COOKIE_VARS["myCookie"];
    echo "\$myCookie hat den Wert: $data";
}
else
{
    $expires = time() + 3600;
    setcookie("myCookie", "Hallo PHP!", $expires, "", "", FALSE);
    echo "Cookie wurde gesetzt!";
}
?>
```

Listing 7.14: Das Array $HTTP_COOKIE_VARS

Dieses Skript macht genau dasselbe wie Listing 7.13 aus dem letzten Kapitel. Lediglich die Ausgabe wird hier über das globale Array realisiert.

7.4.6 Mehrere Variablen in Cookies speichern

In der Praxis übernimmt ein Cookie meist mehr als nur die Informationen einer Variablen, um sie zu speichern. Je nach Situation im Skript ist es nötig, die zu speichernden Daten über verschiedene Funktionen zusammenzufassen, damit sie als einfacher String im Cookie gespeichert werden können.

In dieser Situation ist es sehr nützlich, sich an ein paar Funktionen aus den letzten Kapiteln zu erinnern. Wenn die Daten als Arrays vorliegen, dann bietet sich die Funktion `implode()` an, um sie zu einem String zusammenzufassen. Einzelne Variablen können über den Punktoperator zusammengeschlossen werden.

Um die Werte zu einem späteren Zeitpunkt wieder nutzen zu können, müssen die einzelnen Variablen durch eindeutige Trennzeichen miteinander verbunden werden. Mit der Funktion `explode()` können die Daten dann wieder sauber zerlegt werden.

Funktion	Beschreibung
`implode()`	macht aus einem Array ein String
`compact()`	setzt Variablen zu einem Array zusammen
`explode()`	trennt einen String zu einem Array auf
`$str1 . $str2`	der Punktoperator verbindet zwei Strings

Tabelle 7.10: Array und Stringfunktionen

Das folgende Beispiel zeigt, wie die persönliche Begrüßung auf einer Webseite verwirklicht wird. Zusätzlich wird der Timestamp des letzten Besuches gespeichert.

```
<?
if(isset($myCookie))
{
    $liste = explode("§§", $myCookie);
    $name = $liste[0];
    $remain = round (60 - ((time() - $liste[1]) / 60));
    echo "Hallo $name!<br>";
    echo "In $remain Minuten läuft der Cookie ab.";
}
else
```

```
{
    if(isset($name))
    {
     $expires = time() + 3600;
     $zeit = time();
     $value = $name . "§§" . $zeit;
     setcookie("myCookie", $value, $expires, "", "", FALSE);
     echo "Cookie wurde gesetzt!";
    }
    else
    {
     echo "<form action='listing84.php' method='post'>";
     echo "Name:<br><input name='name'>";
     echo "<input type='submit' value='go'>";
     echo "</form>";
    }
}
?>
```

Listing 7.15: Eine persönliche Webseite mit Countdown

Ausgabe:

```
Hallo Dirk!
In 54 Minuten läuft der Cookie ab.
```

Das Skript ist in drei Teile aufgeteilt, von denen jeder durch eine `if()`-Bedingung aufgerufen wird. Im ersten Schritt präsentiert sich dem User ein Formular, in dem er seinen Namen eintragen kann. Diese Daten werden dann wieder an dasselbe Skript geschickt und zusammen mit dem aktuellen Timestamp gespeichert.

Als Trennzeichen für den Cookiestring wird die Zeichenkombination §§ gewählt, anhand der die Funktion `explode()` die Daten wieder herstellt. Der letzte Schritt in diesem Programm ist die Ausgabe des Cookies zusammen mit einem Countdown, der die Minuten bis zum »Tod« des Cookies herunterzählt.

7.4.7 Übungen

Ü

1. Was sind Cookies und welchen Zweck haben sie?
2. Warum haben Cookies diesen Namen?
3. Wie löscht man Cookies manuell?
4. Warum könnte ein User Cookies ablehnen?

7.5 Sessions

Seit der Version 4 des PHP-Interpreters ist PHP in der Lage selbstständig Sessions einzusetzen und zu verwalten. Diese Technik kann man ohne Zweifel zur Königsklasse der dynamischen Webseitenverwaltung zählen und sie bietet sowohl dem User als auch dem Programmierer eine breite Palette von Möglichkeiten.

7.5.1 Was sind Sessions?

Der Einsatz von Sessions erlaubt es, mit PHP verschiedene Anfragen von ein und demselben Client (Browser) zueinander in Bezug zu setzen. Das Internetprotokoll hat von Haus aus nicht die Möglichkeit, Anfragen miteinander zu vergleichen und festzustellen, welcher Request von welchem Browser kommt.

Die einzige Möglichkeit wäre ein Vergleich der jeweiligen IP-Adressen, um den Benutzer wiederzuerkennen. Allerdings ist die Fehlerquote eines solchen System recht hoch, da IP-Adressen in den meisten Fällen dynamisch zugeteilt werden und den User wechseln können.

Das PHP-Skript weiß also nicht, ob eine eben gestartete Anfrage vom selben User kommt oder ob sich inzwischen ein anderer Surfer mit dem Server in Verbindung gesetzt hat. Dieses zustandslose Verhalten des Protokolls ist in einigen Bereichen der Netzprogrammierung natürlich sehr hinderlich, da gespeicherte Informationen nur schwer einem Benutzer zugeordnet werden können.

Gerade bei größeren Informationen, die nicht über Cookies ausgelagert werden können (sofern der User das überhaupt zulässt), ist es wünschenswert, eine durchgehende Identifikation des Users zu ermöglichen.

Klassische Beispiele für den Einsatz von Sessions sind Internetcommunities oder der inzwischen oft beschworene Webshop. Während der User auf den Seiten des Internetangebotes surft, lädt er immer neue Seiten vom Server herunter. Im Normalfall würde das Skript nach jeder Anfrage alle Daten vergessen und den Benutzer als neuen Besucher einstufen.

Mit Hilfe des Sessionmanagements ist ein Server in der Lage scheinbar unzusammenhängende HTTP-Anfragen zueinander in Bezug zu setzen und die verschiedenen User zu trennen. Dazu verwendet PHP eine zufäl-

lig generierte Session-ID, die jedem User einmalig zugeordnet wird. Anhand dieser ID kann jede Anfrage identifiziert werden und alle relevanten Daten stehen dem Skript wieder zur Verfügung.

Je nach Konfiguration und Usereinstellungen wird die Session-ID entweder als Cookie übergeben oder über die GET-Methode weitergereicht. Das Schöne an dieser Technik ist, dass PHP die komplette Organisation übernimmt, ohne dass man sich um die technischen Hintergründe zu kümmern braucht.

Alle Sessiondaten werden vom PHP-Interpreter in einem temporären Verzeichnis auf dem Server gespeichert und automatisch wieder gelöscht, wenn die Session abgelaufen ist. Die Userdaten werden über eine verschlüsselte Session-ID identifiziert. Es besteht also keine Gefahr, dass die Daten einfach ausgelesen werden können.

Genau wie bei Cookies oder Formularen stehen die Sessiondaten automatisch als Variablen im Skript zur Verfügung. PHP prüft bei jedem Aufruf, ob eine gültige Session exsitiert. Wenn ja, werden alle Daten ins Skript geladen, wenn nicht, wird automatisch eine neue Session angelegt.

7.5.2 Wo werden Sessions eingesetzt?

Sessions können überall da eingesetzt werden, wo ein Surfer über mehrere Seiten hinweg identifiziert werden muss. Gerade auf personalisierten Seiten im Web bietet es sich an, mit Sessions zu arbeiten, da Daten auf diesem Wege sehr einfach verwaltet werden können.

Internetcommunities

Große und kleine Communities im Netz haben die Aufgabe, Menschen zusammenzuführen und Kommunikationswege zu öffnen. Damit jeder User auf der Seite identifiziert werden kann, müssen seine Daten bei jeder neuen Serveranfrage übergeben werden.

Diese Aufgabe wird automatisch von einem geeigneten Sessionmanagement übernommen, so dass alle wichtigen Daten zur Verfügung stehen.

Einkaufen im Netz

Ein weiterer wichtiger Punkt für Sessions ist das Segment der Webshops im Netz. Gerade in Bezug auf Datensicherheit ist es unumgänglich, dass jeder User eindeutig identifiziert werden kann. Gleichzeitig besteht die

Möglichkeit, den Einkaufskorb zu speichern und sicher zu übergeben, damit das Programm an Ende des Einkaufsbummels noch weiß, was der Kunde alles gekauft hat.

Selbst wenn der Einkauf abgebrochen wird, stehen die Daten (je nach Konfiguration) auch noch nach einiger Zeit zur Verfügung.

7.5.3 Sessions mit PHP

PHP bietet verschiedene Möglichkeiten die Sessionverwaltung zu konfigurieren. In den folgenden Kapiteln werde ich die Standardkonfiguration von PHP besprechen. Für das weiterführende Thema rund um die Konfiguration von PHP sei hiermit auf das Kapitel »Die Arbeitsumgebung« verwiesen.

Der Einsatz von Sessions in PHP ist sehr einfach geregelt und kann über einige wenige Funktionen gehandhabt werden. Die komplizierten Hintergründe werden von PHP wunderbar gekapselt, so dass der User und der Programmierer nur mit den nötigen Funktionen und Daten in Berührung kommen.

In jedem Skript, in dem Sessions verwendet werden sollen, muss dies über den Aufruf einer bestimmten Funktion mitgeteilt werden. Dies geschieht entweder über die Funktion `session_start()` oder implizit über das Registrieren einer neuen Variable mit `session_register()`.

In jedem Fall stellt PHP fest, ob schon eine Session existiert oder nicht. Je nach Situation wird dann entweder eine neue Session angelegt oder es werden die Daten der vorhandenen Session geladen.

Um eine Variable für die aktuelle Session benutzen zu können, muss sie registriert werden. Dies geschieht über die Funktion `session_register()`, die als Argument den Namen der Variablen übernimmt. Jede registrierte Variable steht automatisch auf jeder neuen Seite zur Verfügung, auf der die Session gestartet wird.

Um einen Wert wieder aus der Session zu entfernen, muss er über die Funktion `session_unregister()` gelöscht werden. Auch hier wird einfach der Variablenname übergeben.

Das Ende einer Session wird mit der Funktion `session_destroy()` eingeläutet, welche die aktuelle Session sowie alle dazugehörigen Werte aus dem Speicher entfernt. Jeder weitere Aufruf einer Session wird wieder als neue Session gewertet.

Neben diesen elementaren Steuerfunktionen stehen einige Prüffunktionen zur Verfügung, die es erlauben, Informationen über die aktuelle Session zu erhalten. Sie können beispielsweise den Namen des Session ausgeben oder die vorhandenen Variablen auf Gültigkeit überprüfen.

Funktion	Beschreibung
`session_start()`	startet eine Session im Skript
`session_destroy()`	löscht aktuelle Session und alle Daten
`session_name()`	gibt den Namen der Session zurück
`session_save_path()`	gibt den Speicherpfad auf dem Server der Session zurück
`session_id()`	gibt die Session-ID zurück
`session_register()`	registriert eine Variable in der Session
`session_unregister()`	löscht eine Variable aus der Session
`session_unset()`	löscht alle Variablen aus der Session
`session_is_registered()`	überprüft eine Variable, ob sie in der Session gespeichert ist. Gibt TRUE oder FALSE zurück
`session_encode()`	codiert alle Daten der aktuelle Session und gibt sie als String zurück
`session_decode()`	decodiert den String wieder

Tabelle 7.11: PHP-Sessionfunktionen

Die Funktionen `session_register()` und `session_unregister()` sowie die Funktion `session_is_set()` werden mit dem Namen der jeweiligen Variable als Argument gestartet. Es ist nicht zulässig, dass die Variable selbst übergeben wird, da PHP nur den Wert übergeben würde.

```
$var = 128;
session_register("var");
if (session_is_set("var"))
```

```
{
    session_unregister("var");
}
```

Der Name der Variablen muss als String übergeben werden, damit PHP den Wert korrekt speichern kann.

Die Funktionen `session_encode()` und `session_decode()` stimmen im ersten Augenblick ein wenig nachdenklich, da sie für den eigentlichen Sessionvorgang in PHP nicht benötigt werden. Sie erlauben es allerdings, die Daten, die im Laufe einer Session registriert wurden, in einen handlichen String zu verpacken, um sie beispielsweise in einer Datenbank zu speichern.

Mit der Funktion `session_encode()` kann die Codierung wieder aufgehoben werden.

Übergabe der Session-ID

Die Standardkonfiguration für das Sessionmanagement unter PHP legt die Übergabe der Session-ID mit einem Cookie fest. Je nach Konfiguration des Servers kann sich die Sachlage allerdings auch ändern. Außerdem ist es möglich, dass der User das Setzen von Cookies verbietet und man so gezwungen ist auf Alternativen auszuweichen.

Sollte der Server Cookies zulassen, wird ein entsprechend konfigurierter Browser folgende Meldung ausspucken:

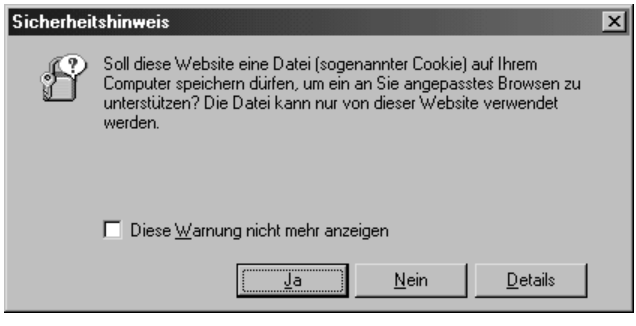


Abbildung 7.11: Eine Session wird als Cookie gespeichert

Sollten Cookies allerdings nicht möglich sein, versucht das PHP-Session-Modul die ID auf anderem Wege zu übertragen. Je nach Einstellung kann das über GET- oder POST-Methode geschehen, die den Interpreter dazu veranlasst, alle Links im Skript automatisch mit der korrekten ID zu versehen.

In der Regel muss sich der Programmierer nicht um die Übergabe der Session-ID kümmern, da das Session-Modul von PHP sehr intelligent auf die meisten Situationen reagiert. Wird der Cookie verweigert, dann übergibt er die Session als URL-Anhang oder nutzt einfach ein Formular für die Daten.

Möchte man sichergehen und die Session-ID manuell übergeben, dann stellt PHP die Sessionkonstante SID zur Verfügung. Sie kann wie jeder andere Wert als Formularvalue oder Linkanhang übergeben werden und wird automatisch als Session-ID erkannt.

```
echo "<a href='script.php?" . SID . "'>Link</a>";
```

PHP ersetzt die Konstante durch den Schlüssel-Wert-String:

```
PHPSESSID = [nummer]
```

Der Wert [nummer] entspricht dabei der dynamisch erzeugten Session-ID des PHP-Interpreters.

Ein Beispiel

Da das wahre Potenzial des Sessionmanagements erst beim Einsatz auf einigen zusammenhängenden Seiten sichtbar wird, besteht das folgende Beispiel aus mehreren Skripten, die die Daten untereinander austauschen. Alle Skripte müssen im selben Verzeichnis auf dem Server liegen, damit die Verlinkung funktioniert.

Es werden im Laufe des Skripts einige Daten gesammelt, die über das Sessionmanagement immer verfügbar sind. Das Ziel ist eine Art »Minicommunity« zu erschaffen, in der der User Daten über sich selbst ablegen kann. Dass die User untereinander nicht kommunizieren können, soll uns zum jetzigen Zeitpunkt nicht stören.

Das erste Listing ist eine Art Login-Seite, auf der der User seinen Namen angeben muss. Gleichzeitig wird der aktuelle Timestamp der Systemzeit gespeichert.

```
<?
// Session wird gestartet!
session_start();
// Ausgabe der Daten!
echo "Sie haben die Session-ID: " . session_id();
echo "<br>";
echo "Ihr Sessionname lautet: " . session_name();
echo "<br>";
// Startzeitpunkt wird festgestellt und registriert!
$startzeit = time();
session_register("startzeit");
?>
<!-- HTML-Formular für den Usernamen -->
Willkommen im Internet ;-)<br>
<Form action="listing85b.php" method="post">
Bitte Name eingeben:<br>
<input name="name">
<input type="submit" value="anmelden">
</form>
<br>
<?
// Sessiondaten werden überprüft!
if (session_is_registered("startzeit"))
{echo "Ihre Startzeit wurde erfolgreich registriert!"; }
?>
```

Listing 7.16: Die Session beginnt …

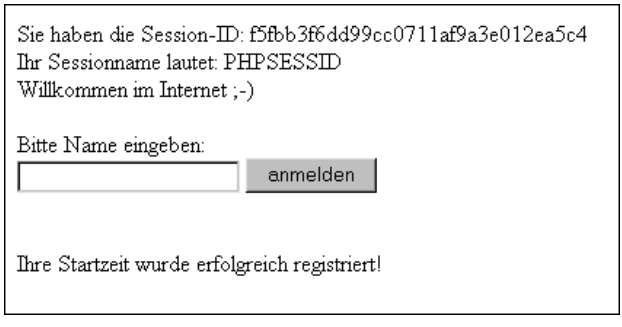


Abbildung 7.12: Der Session erster Teil

Das Skript erschafft eine neue Session und registriert als ersten Wert den Timestamp der aktuellen Systemzeit. Gleichzeitig werde alle Daten rund um diese Session ausgegeben. Der User muss seinen Namen angeben, um auf die nächste Seite zu gelangen.

```
<?
// Session wird gestartet! Alle Daten sind wieder vorhanden!
session_start();
//Formulardaten werden überprüft!
if($name == "") {$name = "nobody";}
// Begrüßung
echo "Hallo $name!<br>";
// Zeit auf der Seite wird berechnet und ausgegeben!
$dzeit = time() - $startzeit;
echo "Sie sind seit $dzeit Sekunden angemeldet!<br><br>";
// Daten werden in der Session registriert, wenn nötig!
if (!(session_is_registered("name")))
{ session_register("name"); }
// Ausgabe weiterer Sessiondaten!
if(session_is_registered("lang"))
{ echo "Sie mögen die Sprache $lang!<br><br>"; }
?>
<!-- HTML-Linkliste -->
<a href="listing85c.php?mylang=C">Ich mag C!</a><br>
<a href="listing85c.php?mylang=PHP">Ich mag PHP!</a><br>
<a href="listing85c.php?mylang=Java">Ich mag Java!</a><br>
<a href="listing85d.php">Session verlassen!</a>
```

Listing 7.17: Die Hauptseite der »Community«

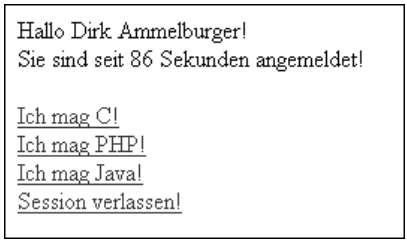


Abbildung 7.13: Der User ist angemeldet

Die nächste Seite beginnt wieder mit dem Aufruf der Funktion `session_start()`, die aber diesmal keine Session anlegt, sondern die Daten der aktuellen Session reaktiviert. Jetzt steht uns die Variable `$startzeit` wieder zur Verfügung, die wir dazu benutzen, die Aufenthaltsdauer des Users zu berechnen.

In einem weiteren Schritt wird der eingegebene Name überprüft und ebenfalls in der Session registriert. Wurde kein Name eingegeben, dann bekommt der User automatisch den Namen »nobody« zugewiesen.

Das Programm überprüft im nächsten Schritt die Session auf die Variable $lang, die allerdings noch nicht registriert wurde. Das ändert sich aber, wenn der User einen der folgenden HTML-Links anklickt. Je nach Link wird eine andere Vorliebe für eine Programmiersprache übergeben.

```
<?
// Session wird gestartet! Alle Daten sind wieder vorhanden!
session_start();
//Begrüßung
echo "Hallo $name!<br>";
// Zeit auf der Seite wird berechnet und ausgegeben!
$dzeit = time() - $startzeit;
echo "Sie sind seit $dzeit Sekunden angemeldet!<br><br>";
// Variablen werden gesetzt!
$lang=$mylang;
// Die Session wird überprüft und bei Bedarf aktualisiert!
if (!(session_is_registered("lang")))
{
      session_register("lang");
}
echo "Sie haben die Sprache $lang gewählt!<br><br>";
?>
<!-- HTML-Link -->
<a href="listing85b.php">weiter</a>
```

Listing 7.18: Auswahl der Programmiersprache

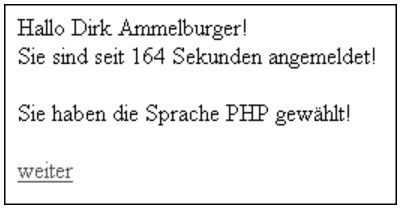


Abbildung 7.14: Die richtige Sprache wurde gewählt

Der erste Teil des Skripts gleicht der vorherigen Seite: Der User wird begrüßt und die Verweildauer auf der Seite berechnet. Danach wird der übergebene Wert ausgewertet und bei Bedarf in der Session registriert. Ein Link führt dann wieder auf die Übersichtsseite der »Mini-Community« zurück.

Da jetzt die Variable `$lang` in der Session registriert ist, gibt das Programm die gewählte Programmiersprache aus. Diese Vorliebe kann jederzeit durch einen Klick auf einen anderen Link geändert werden.

Möchte man die Seiten verlassen, dann genügt ein Klick auf den letzten der vier Links. Er ruft die Logoutseite der »Community« auf.

```
<?
// Session wird gestartet! Alle Daten sind wieder vorhanden!
session_start();
//Begrüßung
echo "Hallo $name!<br>";
// Zeit auf der Seite wird berechnet und ausgegeben!
$dzeit = time() - $startzeit;
echo "Sie waren $dzeit Sekunden auf der Seite!<br><br>";
//Session wird beendet!
session_destroy();
echo "Ihre Session wurde beendet!<br>";
?>
<!-- HTML-Link -->
<a href="listing85a.php">wieder anmelden!</a>
```

Listing 7.19: Die Session wird zerstört

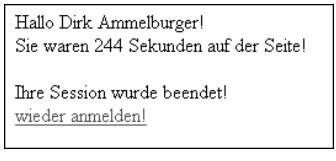


Abbildung 7.15: Das Ende der Session wird verkündet

Das einzig Neue an diesem Skript ist die Zerstörung der Session mit der Funktion `session_destroy()`. Hiermit werden alle gespeicherten Daten sowie die Session-ID aus dem Speicher des Servers gestrichen und eventuelle Cookies gelöscht. Möchte man sich neu anmelden, dann führt ein Link wieder zur Startseite.

Sessions auf Gültigkeit überprüfen

In der Praxis sollte man darauf achten, dass das Programm nur gültige Sessions zulässt. Wenn sich ein User abmeldet oder die Session nach einiger Zeit ausläuft, muss das Programm den Benutzer darauf hinweisen. Der einfachste Weg festzustellen, ob eine Session noch gültig ist, ist die Überprüfung eines Wertes.

```
session_start();
if (!(session_is_registered("name"))) {die "Session abgelaufen!";}
```

Sollte der Wert nicht gesetzt sein, dann kann man davon ausgehen, dass die Session nicht mehr gültig ist. Der User muss sich in einem solchen Fall einfach neu anmelden, damit er eine gültige ID zugewiesen bekommt.

Im Beispielprogramm wurde aus Gründen der Übersicht auf diese Kontrolle verzichtet. Das Ergebnis einer ungültigen Session-ID sieht etwa so aus:

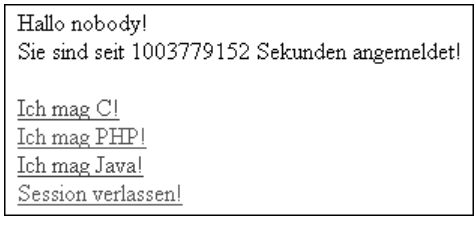


Abbildung 7.16: Eine ungültige Session

Da die Variablen nicht mehr gesetzt sind, gibt das Programm sehr merkwürdige Werte aus. Eine kurze Überprüfung hätte das verhindert.

7.5.4 Übungen

1. Was sind Sessions?
2. Welchen Zweck haben Sessions?
3. Was ist eine Session-ID?
4. Warum muss in diesem Skript die Session nicht gestartet werden?

```
<?
$var = "Hallo PHP!";
session_register("var");
echo $var;
?>
```

5. Wie kann die Session-ID übergeben werden?
6. Warum sollte man eine Session immer auf Gültigkeit überprüfen?

8 Strings und dynamisches HTML

PHP bietet eine umfangreiche Sammlung von Operatoren und Funktionen rund um die Stringbearbeitung an. Einige dieser Möglichkeiten wurden bereits im ersten Teil des Buches ausführlich besprochen. Besonders Funktionen zur Stringmanipulation sind in der Praxis von großer Bedeutung, da PHP in den meisten Fällen mit Text und nicht mit Zahlenwerten umgehen muss.

Folgende Möglichkeiten haben Sie bisher kennen gelernt:

- Funktionen zum Ersetzen von Zeichen und Zeichenketten
- Funktionen zum Umwandeln von Zeichenketten
- Verbindungsfunktionen und Operatoren
- Funktionen zum gezielten Trennen von Zeichenketten
- Suchfunktionen für Strings
- Vergleichsfunktionen für Strings
- Formatierte und unformatierte Ausgabe von Strings

Das folgende Kapitel wird sich mit einem weiteren Teil der Stringfunktionen beschäftigen, der speziell für den Umgang mit HTML-Code geschaffen wurde. Gerade der Umgang mit Sonderzeichen und die formatierte Ausgabe von Webseiten ist oft problematisch. PHP bietet für diese Situationen eine Palette von Werkzeugen, die jedem Entwickler die Arbeit sehr erleichtern.

Ein weiterer Bereich dieses Kapitels setzt sich mit der Erstellung von dynamischem HTML auseinander. Als Anwendungsbeispiel soll ein ewiger Kalender erstellt werden, der als HTML-Tabelle realisiert wird und unbeschränkte Gültigkeit hat.

Der wahrscheinlich wichtigste Teil dieses Kapitels ist die Einführung in den Bereich der regulären Ausdrücke. Diese ursprünglich von UNIX stammende Technik gehört zu den mächtigsten Werkzeugen im Umgang mit Texten.

Der letzte Teil des Kapitels behandelt die servereigenen Umgebungsvariablen. Diese speichern alle relevanten Informationen über die aktuelle HTTP-Anfrage und stellen sie dem Programmierer zur Verfügung. Wer also mehr über seine User erfahren möchte, wird um diese Informationsquelle nicht herumkommen.

8.1 Kapitelübersicht

- HTML-Stringfunktionen
- Dynamisches HTML
- Reguläre Ausdrücke
- Umgebungsvariablen

8.2 HTML und Stringfunktionen

Die folgende Funktionen unterstützen den Programmierer bei seiner Arbeit mit HTML-Texten und URL-Angaben. Für beide Bereiche stehen einige Funktionen zur Verfügung, die universell einsetzbar sind und Probleme elegant lösen.

8.2.1 HTML-Funktionen

Ein Text, der durch HTML-Anweisungen formatiert wurde, unterscheidet sich von einem normalen Fließtag durch eine Reihe von Sonderzeichen und Anweisungen, die die Lesbarkeit des Textes außerhalb des Browsers stark einschränken. Auf der anderen Seite ist es notwendig, eine ganze Menge von Konventionen im Umgang mit Texten für das Internet einzuhalten, damit die Informationen auch in jedem Browser angezeigt werden können.

PHP bietet für beide Richtungen Funktionen an, welche die Codierung und Decodierung von Texten automatisieren. Gleichzeitig ist es möglich, Sonderzeichen, die in HTML eine besondere Bedeutung haben (<, >, & etc.), durch Codestrings zu maskieren, damit der Browser sie nicht interpretieren kann. So kann man beispielsweise HTML-Code als Fließtext im Browser darstellen.

Zusätzlich bietet PHP Funktionen an, die speziell für HTML-Dateien entworfen wurden. Sie erlauben es, Informationen über die Datei zu erhalten.

Funktion	Beschreibung
`htmlspecialchars($str)`	Konvertiert HTML-Sonderzeichen in Maskierungscodes, um die Interpretation zu unterbinden.
`htmlentities($str)`	Wie `htmlspecialchars()`, allerdings inklusive Sonderzeichen und Umlaute.
`get_meta_tags($datei)`	Gibt ein Array mit den Metatags einer HTML-Datei zurück.
`nl2br($str)`	Wandelt Zeilenumbrüche in ` `-Tags um.
`strip_tags($str)`	Entfernt HTML- und PHP-Tags.
`wordwrap($str, $int, $str2)`	Bricht Zeilen automatisch nach einer bestimmten Anzahl von Buchstaben um.

Tabelle 8.1: HTML-Funktionen

Zu jeder Funktion wird auf den folgenden Seiten ein Beispiel besprochen. Die Ein- und Ausgabe wird jedes Mal dokumentiert.

htmlspecialchars() und htmlentities()

Die beiden wichtigsten Werkzeuge in dieser Gruppe sind die Funktionen `htmlspecialchars()` und `htmlentities()`. Beide haben die Aufgabe alle HTML-Sonderzeichen im übergebenen String so zu konvertieren, dass der Browser sie nicht mehr interpretiert, sondern als Text ausgibt.

Der Unterschied zwischen den beiden Funktionen liegt versteckt im Detail: Die Funktion `htmlspecialchars()` konvertiert alle Sonderzeichen, die direkt mit der HTML-Codierung zusammenhängen. Betroffen sind alle eckigen Klammern < > sowie das Zeichen & und alle Anführungszeichen.

Das folgende Beispiel zeigt die Funktion im Einsatz:

```
<?
$str = "<table width='200' border='1'> <tr> <td> <b> Titel </b> </td>
<td> <b> Autor </b> </td> </tr> <tr> <td> JavaScript </td> <td> K.
Koch </td> </tr> <tr> <td> GoTo PHP4 </td> <td> D. Ammelburger </td>
</tr> <tr> <td> C++ </td> <td> J. Liberty </td> </tr> </table>";
$str2 = htmlspecialchars($str);
echo $str2;
?>
```

Listing 8.1: Die Funktion htmlspecialchars()

Ausgabe:

```
<table width='200' border='1'> <tr> <td> <b> Titel </b> </td> <td>
<b> Autor </b> </td> </tr> <tr> <td> JavaScript </td> <td> K. Koch </
td> </tr> <tr> <td> GoTo PHP4 </td> <td> D. Ammelburger </td> </tr>
<tr> <td> C++ </td> <td> J. Liberty </td> </tr> </table>
```

Der Browser sollte normalerweise eine Tabelle mit Büchern und Autorennamen ausgeben. Aber Dank der Funktion `htmlspecialchars()` wurde die komplette HTML-Codierung aufgehoben und durch andere Strings ersetzt.

Schaut man sich den HTML-Code genauer an, dann stellt man fest, dass sich PHP einiger Sonderzeichen bedient hat, um die HTML-Zeichen zu maskieren. Hier eine Tabelle mit allen Maskierungsstrings:

HTML-Zeichen	Codierung
<	`<`
>	`>`
„	`&quot;`
&	`&`

Tabelle 8.2: HTML-Maskierung

Der Browser erkennt die Maskierung und stellt die Zeichen wieder so dar, wie sie ursprünglich im String aufgetreten sind, ohne sie allerdings zu interpretieren.

Die Funktion `htmlentities()` macht im Prinzip genau dasselbe, die `htmlspecialchars()`. Diese Funktion geht aber noch einen Schritt weiter und maskiert nicht nur HTML-Sonderzeichen, sondern auch Umlaute und weitere sprachspezifische Sonderzeichen.

```
<?
$str = "Ä Ö Ü ä ö ü ß";
$str2 = htmlentities($str);
echo $str2;
?>
```

Listing 8.2: Die Funktion htmlentities()

Ausgabe:

```
Ä Ö Ü ä ö ü ß
```

Wie erwartet ist die Bildschirmausgabe unverändert. Ein Blick in den HTML-Code zeigt, was die Funktion gemacht hat:

```
&Auml; &Ouml; &Uuml; &auml; &ouml; &uuml; &szlig;
```

Alle Umlaute und Sonderzeichen wurden durch Maskierungsstrings ersetzt, die der Browser wieder als den jeweiligen Buchstaben interpretiert. Der Einsatz dieser Maskierung ist für internationale Webseiten sehr wichtig, damit Texte auch auf ausländischen Browsern korrekt dargestellt werden. Die Sonderzeichenstrings sorgen dafür, dass alle Zeichen in der Form dargestellt werden, wie es geplant war.

get_meta_tags()

Die Funktion `get_meta_tags()` erlaubt es, die Meta-Tags einer HTML-Datei auszulesen und in einem Programm auszuwerten. Das Ergebnis dieser Auswertung ist ein assoziatives Array, das alle Meta-Tags der Seite speichert. Der Schlüssel ist jeweils der Name des Meta-Tags und der Wert der jeweilige Inhalt.

Als Parameter wird der Funktion entweder ein Dateiname oder eine URL übergeben. Im Beispiel soll eine HTML-Datei ausgelesen werden, die folgenden Inhalt hat:

```
<html>
<head>
<meta name="description" content="Beispielseite für PHP">
<meta name="author" content="Dirk Ammelburger">
<meta name="keywords" content="meta, tags, Beispiel, PHP, Strings">
</head>
<body>
Dummypage
</body>
</html>
```

Listing 8.3: HTML-Seite mit Meta-Tags

Diese Datei stellt drei Meta-Tags zur Verfügung, die den Inhalt der Webseite beschreiben. Die Meta-Tags müssen untereinander stehen, damit die Funktion `get_meta_tags()` sie erkennt. Sollte ein weiterer Meta-Tag in derselben Zeile folgen, wird er ignoriert.

Das Beispielprogramm hat folgenden Aufbau:

```
<?
$array = get_meta_tags("index.html");
foreach($array as $key => $element)
{
echo "$key: $element<br>";
}
?>
```

Listing 8.4: get_meta_tags()

Ausgabe:

```
description: Beispielseite für PHP
author: Dirk Ammelburger
keywords: meta, tags, Beispiel, PHP, Strings
```

Alternativ zu einer Datei auf dem Server kann auch eine URL als Parameter für die Funktion `get_meta_tags()` angegeben werden. Der PHP-Interpreter startet automatisch einen HTTP-Request und übergibt die zurückgegebenen Daten an die Funktion.

Diese Möglichkeit ist besonders für Suchmaschinen sehr praktisch, weil Neuanmeldungen sofort überprüft werden können.

```
<?
$array = get_meta_tags("http://www.kulturbrand.de/index.html");
foreach($array as $key => $element)
{
echo "$key: $element<br>";
}
?>
```

Ausgabe:

```
language: de
keywords: download,freeware,script,cgi,bilder,kulturbrand,black-
board,knowware,perl,geld, C++, C, Windows, Programmierung, Webde-
sign, GUI, Java, Webmaster, Ammelburger, Unix, Linux, PHP
revisit-after: 01 month
description: Programmierung und Webdesign auf kulturbrand.de . C++,
Perl, CGI, Java, PHP, Windows und vieles mehr...
```

Im Skript gibt es keinen Unterschied, außer die Angaben im Funktionskopf.

strip_tags()

Die Funktion `strip_tags()` ist eine der wichtigsten Funktionen im Umgang mit HTML-Texten. Sie erlaubt es, die komplette HTML-Codierung aus einem String zu entfernen, ohne dabei den Inhalt des Textes zu verändern. Dies war vorher nur mit komplizierten regulären Ausdrücken möglich.

```
<?
$str="<font size='5' color='green'><b>Hallo PHP</b></font>";
echo $str;
echo "<br>";
echo strip_tags($str);
?>
```

Listing 8.5: strip_tags()

Abbildung 8.1: Vorher und nachher …

Der Übergabeparameter ist der String, der von seiner HTML-Fomatierung befreit werden soll. Das Skript zeigt den Unterschied zwischen einem Text, der HTML-formatiert ausgegeben wird, und denselben Text nach der »Behandlung«.

Schaut man sich den HTML-Code an, dann wird deutlich, dass nichts außer dem »Hallo Welt«-String übrig geblieben ist.

nl2br()

Zeilenumbrüche sind ein Thema, über das man ganze Bücher füllen könnte. In einem Fließtext und in PHP-Strings werden Zeilenumbrüche mit dem Sonderzeichen `\n` markiert, damit das Betriebssystem weiß, dass hier eine neue Zeile beginnt.

Die HTML-Codierung sieht für Zeilenumbrüche den Tag `<BR>` vor und kümmert sich nicht um Formatierungen im Quellcode. Das ist auch gut so, denn sonst würde die Erstellung einer Webseite schnell im Chaos enden.

Trotz allem bleibt die Konvertierung von Zeilenumbrüchen ein Problem, das vor allem bei mehrzeiligen Eingabefeldern in Formularen auftritt. Möchte man den Text aus diesen Feldern exakt so übernehmen, wie der User ihn eingegeben hat, dann wird es schwierig. Dieses Problem löst die Funktion `nl2br()`.

Das Beispiel zeigt eine Webseite mit einer Textarea, deren Inhalt nach dem Abschicken wiedergegeben werden soll.

```
<form action="listing91.php" method="post">
<textarea name="str" cols="28" rows="8"></textarea>
<input type="submit" value="go">
</form>
<?
echo $str;
echo "<br><br>";
$format_str = nl2br($str);
echo $format_str;
?>
```

Listing 8.6: Die Funktion nl2br()

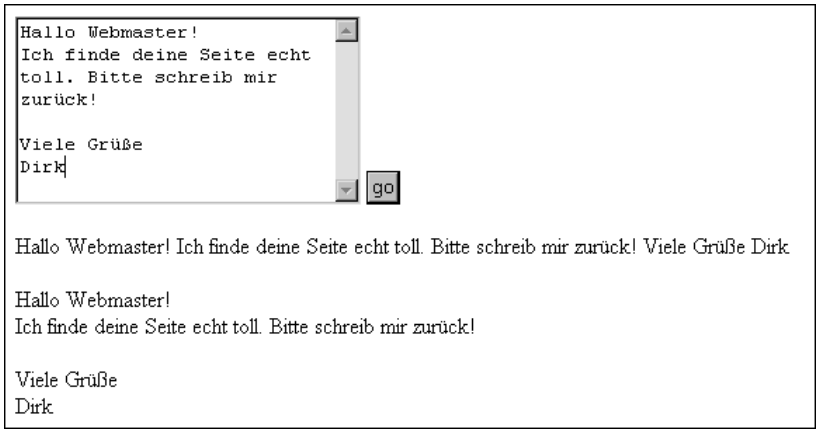


Abbildung 8.2: Ein Feedbackformular

Die Abbildung zeigt deutlich, was die Funktion `nl2br()` bewirkt. Sie stellt die Formatierung, wie sie der User im Formular angegeben hat, in HTML wieder her. Alle Zeilenumbrüche werden durch ein `<br>` ersetzt, wie ein Blick in den HTML-Quellcode zeigt.

```
Hallo Webmaster!<br>Ich finde deine Seite echt toll. Bitte schreib
mir zurück!<br><br>Viele Grüße<br>Dirk
```

Die Funktion `wordwrap()` bewegt sich in ähnlichen Gefilden, hat aber einen anderen Hintergrund.

wordwrap()

Die Funktion trennt nach einer angegebenen Anzahl von Zeichen einen String mit Hilfe eines Trennzeichens auf. Dabei wird nie innerhalb eines Wortes getrennt, sondern immer zwischen zwei Worten. Auf diese Weise ist es möglich, einen Text sauber auf eine bestimmte Breite zu formatieren, ohne die Struktur anzugreifen.

Wird kein Trennzeichen angegeben, dann verwendet die Funktion automatisch das Zeichen `\n` für einen Zeilenumbruch. Soll der Text in einem Browser ausgegeben werden, dann muss man entweder danach die Funktion `nl2br()` verwenden oder gleich das Trennzeichen `<br>` in die Funktion einsetzen.

```
<?
$str = "Und eines Tages hatten sie es satt, für andere Geld reinzuho-
len und dunkle Limousinen zu fahren. Sie wollten keine Sonnenbrillen
mehr tragen und elegante Lederkoffer an geheimen Orten übergeben, um
die Welt vor neuen Kriegen zu retten. Vorbei waren die Zeiten von
durchfeierten Nächten und luxuriösen Jachten, auf denen immer neue
Schönheiten auf sie warteten um sich zu vergnügen. Sie wollten end-
lich selbstständig sein und auf eigenen Beinen stehen. Es war genau
die richtige Zeit für eine Änderung. Und als die Chance kam, ergrif-
fen sie diese. Eine neue Zeit brach an...";
$wrap_str = wordwrap($str, 45, "<br>");
echo $wrap_str;
?>
```

Listing 8.7: Zeilenumbruch mit wordwrap()

Die Anzahl der Zeichen ist eine Maximalangabe. Das heißt, die Funktion wird versuchen diese Anzahl nicht zu überschreiten, um die Formatierung einzuhalten. Wörter, die mehr Zeichen haben, sind die Ausnahme. Ein Wort gilt immer als beendet, wenn ein Leerzeichen erscheint.

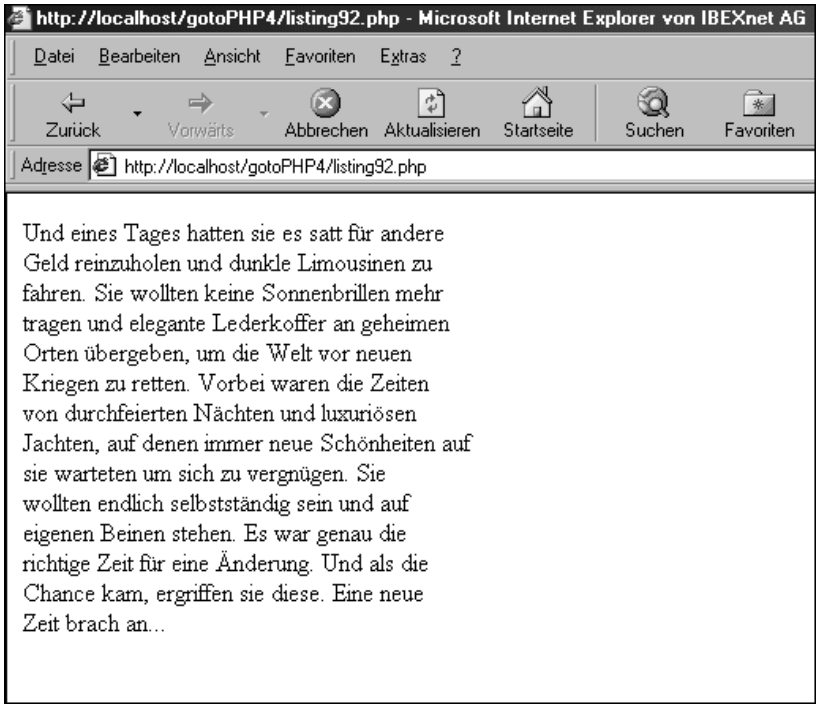


Abbildung 8.3: Prosa schön formatiert …

8.2.2 URL-Funktionen

Die Funktionen aus diesem Kapitel werden benötigt, um mit Hilfe der GET-Methode Daten URL-kodiert zu übergeben. Im Kapitel über dynamische Webseiten wurde die GET-Methode bereits ausführlich besprochen. Das wesentliche Merkmal dieser Technik ist die Möglichkeit Daten an die URL der Webseite anzuhängen und sie so einem Skript verfügbar zu machen.

Diese Daten werden in der Form von Schlüssel-Wert-Paaren zu einem String zusammengefasst und jeweils durch das &-Zeichen getrennt. Um Fehler bei der Übertragung der Daten zu vermeiden, werden Leerzeichen, sprachliche Besonderheiten (Ä, Ö, Ü, ß etc.) und alle anderen Sonderzeichen in der URL verschlüsselt.

Die folgenden Funktionen erlauben es, Strings in ein URL-taugliches Format zu konvertieren oder wieder in einen lesbaren String umzuwandeln.

Funktion	Beschreibung
`rawurlencode($str)`	codiert einen String in ein URL-taugliches Format
`rawurldecode($str)`	entschlüsselt einen String, der mit `rawurlencode()` verschlüsselt wurde
`parse_str($str)`	zerlegt einen URL-String in seine Variablen

Tabelle 8.3: URL-Funktionen in PHP

Die Codierung von Sonderzeichen für ein HTTP-Request wird über ein recht einfaches Schema realisiert. Alle Zeichen, die den Aufbau der URL stören könnten, werden durch ein Prozentzeichen ersetzt und durch den hexadezimalen Wert des ASCII-Systems verschlüsselt. Leerzeichen haben zum Beispiel den Wert %20 und Fragezeichen den Wert %3F.

```
<?
$str="Dieser String soll für eine URL codiert werden!";
echo rawurlencode($str);
?>
```

Listing 8.8: Ein String wird codiert

Ausgabe:

```
Dieser%20String%20soll%20f%FCr%20eine%20URL%20codiert%20werden%21
```

Die Decodierung geschieht auf demselben Wege und ist genauso einfach zu handhaben.

```
<?
$str="Die-
ser%20String%20soll%20f%FCr%20eine%20URL%20codiert%20werden%21";
echo rawurldecode($str);
?>
```

Listing 8.9: Ein String wird decodiert

Ausgabe:

```
Dieser String soll für eine URL codiert werden!
```

Wenn ein URL-String decodiert wurde, ist der nächste logische Schritt das Auslesen der Werte. Für diese Aufgabe hat PHP eine eigene Funktion, die es problemlos erlaubt, einen URL-String zu zerlegen. Dabei wird jede Schlüssel-Wert-Kombination einzeln ausgewertet und in einer Variable

gespeichert. Der Variablenname ergibt sich aus dem Schlüssel des Paares und übernimmt den Wert rechts vom Gleichheitszeichen.

Das Besondere an der Funktion `parse_str()` ist, dass sie keinen expliziten Rückgabewert hat. Typischerweise hätte man ein assoziatives Array erwartet, das die Werte speichert, doch das ist hier nicht der Fall. Die Funktion deklariert einfach für jedes Schlüssel-Wert-Paar eine eigene Variable, die den Wert speichert.

```
<?
$str=$vname=Dirk&nname=Ammelburger&url=kultur-
brand.de&email=dirk@ammelburger.de";
parse_str($str);
echo $vname;
echo "<br>";
echo $nname;
echo "<br>";
echo $url;
echo "<br>";
echo $email;
?>
```

Listing 8.10: Die Funktion parse_str()

Ausgabe:

```
Dirk
Ammelburger
kulturbrand.de
dirk@ammelburger.de
```

Das Listing zeigt, wie ein typischer URL-String über die Funktion `parse_str()` zerlegt wird. Nachdem die Funktion aufgerufen wurde, stehen vier Variablen zur Verfügung, die daraufhin ausgegeben werden.

8.2.3 Übungen

1. Was ist der Unterschied zwischen HTML-Text und normalem Fließtext?
2. Warum werden Sonderzeichen in HTML codiert?
3. Warum werden Sonderzeichen in einer URL codiert?

8.3 HTML dynamisch erzeugen

Der Titel dieses Kapitels mag ein wenig pathetisch klingen, wenn man bedenkt, dass jede Ausgabe eines PHP-Skripts im Prinzip HTML dynamisch erzeugt. Darum wird dieses Kapitel auch grundsätzlich wenig Neues präsentieren, sondern vielmehr das Gelernte weiter vertiefen. Oft ist es nämlich sehr nützlich, große HTML-Seiten nicht von Hand zu programmieren, sondern sie in weiten Teilen dynamisch erzeugen zu lassen.

Ansätze dieser Technik wurden bereits im Kapitel über dynamische Formulare besprochen. Über verschiedene Schleifen wurden Teile eines HTML-Formulars dynamisch erzeugt und ausgegeben, so dass der Arbeitsaufwand für dieses Formular deutlich gesunken ist.

Es bietet sich an, dass dieses Prinzip auf ähnliche Strukturen einer Webseite übertragen wird. Typische Anwendungsmöglichkeiten für den dynamischen Aufbau einer Seite sind HTML-Bereiche, die einem bestimmten Muster folgen. Zuerst fällt einem natürlich der Aufbau einer Tabelle ein, aber es gibt noch weitere Möglichkeiten:

- Tabellen
- Formulare
- Framesets
- Linklisten
- Datensätze mit Trennzeichen

8.3.1 Das Prinzip

Der Aufbau des PHP-Programms basiert im Kern immer auf einer Schleife, die eine bestimmte Anzahl von HTML-Anweisungen ausgibt. In den meisten Fällen ändert sich dabei der Inhalt der Ausgabe, so dass nur die Formatierung beibehalten wird.

```
<?
$arr=array("A","B","C","D","E","F","G","H","I","J","K");
echo "<table border=1>";
foreach($arr as $var)
{
   echo "<tr><td> <b>$var</b> </td>";
   for($i=1;$i<=11;$i++)
```

```
    {
        echo "<td width='20' align='center'> $i </td>";
    }
    echo "</tr>";
}
echo "</table>";
?>
```

Listing 8.11: Dynamische HTML-Tabelle

A	1	2	3	4	5	6	7	8	9	10	11
B	1	2	3	4	5	6	7	8	9	10	11
C	1	2	3	4	5	6	7	8	9	10	11
D	1	2	3	4	5	6	7	8	9	10	11
E	1	2	3	4	5	6	7	8	9	10	11
F	1	2	3	4	5	6	7	8	9	10	11
G	1	2	3	4	5	6	7	8	9	10	11
H	1	2	3	4	5	6	7	8	9	10	11
I	1	2	3	4	5	6	7	8	9	10	11
J	1	2	3	4	5	6	7	8	9	10	11
K	1	2	3	4	5	6	7	8	9	10	11

Abbildung 8.4: Schiffe versenken im Browser

Das Programm sieht auf den ersten Blick komplizierter aus, als es eigentlich ist. Im Prinzip besteht es aus zwei Schleifen, die so ineinander verschachtelt sind, dass sich eine zweidimensionale Struktur ergibt. Die erste Schleife generiert für jedes Arrayelement eine Tabellenreihe und ruft dann eine weitere Schleife auf. Diese füllt die Reihe mit dem eigentlichen Inhalt (in diesem Fall einfach die Zahlen 1 bis 11).

Wenn diese Schleife abgeschlossen wird, kehrt der PHP-Interpreter in die übergeordnete Schleife zurück und schließt die Tabellereihe ab. Der Kreis schließt sich und die Schleife beginnt von vorn.

Das Ergebnis ist eine HTML-Tabelle, die das grundlegende Spielfeld für das bekannte Spiel »Schiffe versenken« darstellt. Jede Reihe ist einem Buchstaben zugeordnet, und jedes Feld hat eine eigene Nummer. So kann über die Koordinaten aus Buchstabe und Nummer jedes Spielfeld anvisiert werden.

Diese Technik der Datenformatierung wird oft in Zusammenhang mit Datenbanken verwendet, da die Datenstruktur dieser Software ebenfalls in Tabellenform vorliegt. Der Aufbau erinnert nicht zufällig an ein Array, die Gemeinsamkeit ist gewollt. Zweidimensionale Arrays sind die Grundlage für die Kommunikation mit einer Datenbank.

8.3.2 Der ewige Kalender

Als kleiner Höhepunkt zu diesem Thema möchte ich hier ein Programm vorstellen, das die Möglichkeiten von dynamischen HTML mit PHP aufzeigt. In diesem Kapitel werden wir auch auf die Grundlagen der vergangenen Kapitel zurückgreifen, um das volle Potenzial von PHP zu nutzen.

Ziel ist, einen Jahreskalender zu programmieren, der komplett mit den vorhandenen Funktionen von PHP realisiert werden soll. Die Darstellung wird praktischerweise in Tabellenform erfolgen und sich über eine Webseite erstrecken. Als zusätzliche Möglichkeit soll das Programm jedes beliebige Jahr dynamisch berechnen können. Schaltjahre werden berücksichtigt.

Diese Voraussetzungen schreien natürlich nach einem exzessiven Einsatz von PHP-Datumsfunktionen, die wir bereits kennen gelernt haben. Und so ist es auch: Besonders die Funktion `checkdate()` hat hier eine tragende Rolle.

```
<?
if(!(isset($jahr))) {$jahr=date("Y",time());}
$timestamp=mktime(1,1,1,1,1,$jahr);

echo "<center><a href='listing96.php?jahr=". ($jahr-1) .">zurück</
a> ";
echo " <a href='listing96.php?jahr=". ($jahr+1) .">weiter</a></cen-
ter>";

echo "<center><font size='5' face='tahoma'>Jahr $jahr</
font><br><br>";
echo "<table width='800' border='0'><tr>";
$monat=1;
$tag=1;
while(checkdate($monat, $tag, $jahr))
{
echo "<td align='center' valign='top'>". date("F",$timestamp);
echo "<table width='130' border=1><tr>";
```

```
    while(checkdate($monat, $tag, $jahr))
    {
        echo "<td>$tag</td>";
        if(($tag%7) == 0) {echo "</tr><tr>";}
        $tag++;
    }
echo "</tr></table></td>";
$monat++;
$timestamp=$timestamp+($tag-1)*24*60*60;
$tag=1;
if($monat==7){echo "</tr><tr>";}
}
echo "</tr></table></center>";
?>
```

Listing 8.12: Der ewige Kalender

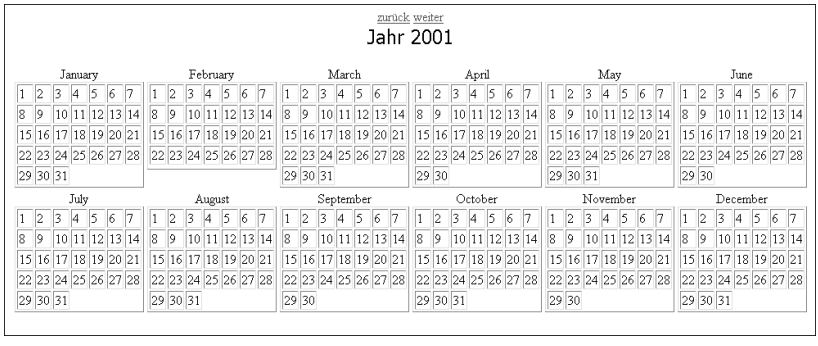


Abbildung 8.5: Der Kalender im Browser

Das Ergebnis dieser knappen Seite Quelltext ist ein übersichtlicher Jahreskalender, der alle Tage nach Monaten anzeigt. Über die Links »weiter« und »zurück« kann man sich durch die Jahre klicken, die jeweils am Anfang der Seite angezeigt werden.

Der Aufbau des Programms folgt im Großen und Ganzen den Grundlagen des Programms für die »Schiffe Versenken«-Tabelle. Nachdem die Variable `$jahr` überprüft (und gegebenenfalls gesetzt) wurde, wird eine Überschrift ausgegeben. Danach folgt das bekannte Muster der verschachtelten Schleifen, die es erlauben, ein zweidimensionales Feld aufzubauen.

Die Abbruchbedingungen der Schleifen werden über die bereits erwähnte Funktion `checkdate()` realisiert. Sie überprüfen, ob ein Datum gültig ist

oder nicht. Bei jedem Schleifendurchlauf werden die Variablen `$tag` und `$monat` inkrementiert, so dass sich eine fortlaufende Ausgabe von Tagen und Monaten ergibt. Stößt die Schleife auf ein ungültiges Datum, ist der jeweilige Monat beendet beziehungsweise das Jahresende erreicht, und die Schleife bricht ab.

Die Ausgabe der Monatsnamen wurde über die Funktion `date()` realisiert, die es erlaubt, englische Bezeichnungen aus einem Timestamp abzuleiten. Damit auch der jeweils aktuelle Timestamp vorliegt, wird der am Anfang berechnete Timestamp vom 1. Januar des Jahres in jedem Durchgang der äußeren Schleife erhöht. Die Anzahl der Sekunden werden ganz einfach über eine Multiplikation errechnet.

Wie Sie vielleicht schon bemerkt haben, kann ich das Versprechen eines »ewigen« Kalenders nicht ganz einhalten. Das Programm funktioniert fehlerfrei zwischen den Jahren 1970 und 2037. Sobald man diese Zone allerdings verlässt, gibt der PHP-Interpreter (je nach Konfiguration) eine Warnmeldung aus, die besagt, dass die Funktion `date()` einen Fehler verursacht hat.

Der Grund für diese Erscheinung ist offensichtlich und wurde im Kapitel über Datum- und Zeitfunktionen schon eingehend besprochen. Der UNIX-Timestamp wird vom 01.01.1970 berechnet und gibt deshalb vor diesem Datum eine Minuszahl aus. Auf der anderen Seite endet die »Timestamp-Ära« nach einer bestimmten Anzahl von Jahren, da die Zahl zu diesem Zeitpunkt einen Überlauf produziert.

Natürlich ist dieses Programm in seinen Möglichkeiten noch nicht erschöpft. Ein sinnvolle Erweiterung wäre zum Beispiel die Einführung von Wochentagen oder die farbliche Hervorhebung von Feiertagen. Ebenfalls denkbar ist eine Art Terminplaner fürs Netz, der es erlaubt, datenbankgestützt Termine zu speichern und zu verwalten.

8.4 Reguläre Ausdrücke

Eines der wichtigsten Merkmale von PHP ist die Unterstützung von regulären Ausdrücken. Diese Fähigkeit, die PHP eindeutig von Perl geerbt hat, erlaubt es, souverän mit Zeichen und Zeichenketten umzugehen und jeder Anweisung exakt definierte Anforderungen zugrunde zulegen.

Ein regulärer Ausdruck (engl.: regular expression) lässt sich am einfachsten als eine Art Mustervorlage beschreiben, die es erlaubt, Übereinstimmungen in Zeichenketten zu finden. Diese Vorlage ist dabei nicht nur auf einfache Buchstaben oder Zeichen angewiesen, sondern kann auch komplexe Bedingungen formulieren wie:

```
"Das erste Zeichen muss eine Zahl sein gefolgt von einem ‚e' oder
‚E'. Danach darf das Wort nur noch zehn Zeichen lang sein und muss
mit ‚.txt' enden."
```

Mit herkömmlichen PHP-Methoden wäre es sehr kompliziert oder fast unmöglich, eine solche Bedingung an einem String zu überprüfen. Reguläre Ausdrücke hingegen machen die Sache relativ einfach.

Ich sage relativ, da reguläre Ausdrücke wie ein Baukastensystem aufgebaut sind. Man hat eine endliche Anzahl von Steinen, von denen jeder für sich eine einfache Aufgabe erfüllt. Aber diese Bausteine können auf unendlich viele Arten zusammengesetzt werden, so dass es oft einiges an Erfahrung braucht, um das gewünschte Resultat zu erhalten.

Trotz allem sind reguläre Ausdrücke nicht schwierig! Sie benötigen einfach eine bestimmte Denkstruktur, die man sich schwer anlesen kann, sondern die auf praktischen Erfahrungen beruht. Nach diesem Kapitel werden Sie einige nützliche Ausdrücke konstruieren können, die in vielen Situationen sehr hilfreich sein werden. Allerdings ist das Thema reguläre Ausdrücke damit nicht abgeschlossen.

Andere Autoren haben ganze Bücher zu diesem Thema geschrieben, die den Stoff natürlich viel intensiver beleuchten, als es mir möglich ist.

8.4.1 Aufbau von regulären Ausdrücken

Der Aufbau eines regulären Ausdrucks richtet sich nach der Komplexität der Anfrage. Je komplizierter eine Fragestellung ist, desto größer und komplexer wird der reguläre Ausdruck für die Lösung. In der Regel wird das gewünschte Suchmuster einfach Stück für Stück zusammengesetzt, bis das Ergebnis stimmt.

Einfache reguläre Ausdrücke

Für eine einfache Anfrage, wie zum Beispiel die Suche nach konstanten Zeichen oder Zeichenketten, ist kein Hintergrundwissen nötig. Das Muster besteht einfach aus dem zu suchenden String.

```
<?
$str = "Dies ist ein Teststring für die Suche nach regulären
Ausdrücken";
$bool = ereg("ring", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.13: Einfache reguläre Ausdrücke

Ausgabe:

```
Gefunden!
```

Die Funktion `ereg()` gehört zu den wichtigsten Funktionen zum Umgang mit regulären Ausdrücken innerhalb der Sprache PHP. Sie sucht in einem String nach einem Muster und gibt je nach Erfolg TRUE oder FALSE zurück.

Das Beispielprogramm hat die einfache Aufgabe, das Muster »ring« im String $str zu finden. Da der String vorhanden ist, gibt das Programm eine Erfolgsmeldung aus. Zugegeben, dieses Beispiel ist nicht sehr aufregend und hätte auch ohne reguläre Ausdrücke gelöst werden können. Der nächste Schritt wird dafür umso spannender.

Viel häufiger als das eben besprochene Beispiel ist eine gewisse Unsicherheit bezüglich des Suchmusters. Wenn Sie nicht genau wissen, wie der zu suchende String aussieht, erlauben Ihnen die regulären Ausdrücke Alternativen zu formulieren. Dies geschieht über das Sonderzeichen | .

```
<?
$str = "Dies ist ein Teststring für die Suche nach regulären
Ausdrücken";
$bool = ereg("ring|Ring", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.14: Suche nach Alternativen

Ausgabe:

```
Gefunden!
```

Ähnlich wie in PHP kann über den senkrechten Strich ein ODER generiert werden, das es erlaubt, zwischen verschiedenen Möglichkeiten zu unterscheiden.

Zeichengruppen

Für die Suche nach bestimmten Zeichen oder Zeichengruppen bieten die regulären Ausdrücke eine Reihe von praktikablen Möglichkeiten. Ein einzelnes Zeichen zu finden ist natürlich nicht weiter schwierig, wenn man sich das letzte Kapitel anschaut. Komplizierter wird es allerdings, wenn man feststellen möchte, ob ein String generell Buchstaben enthält oder nicht. Für solche Fälle gibt es die so genannten Zeichengruppen, die selbst definiert werden können.

Zeichengruppen werden immer in eckige Klammern geschrieben und enthalten die Menge von Zeichen, auf die ein String untersucht werden soll.

```
<?
$str = "Dies ist ein Teststring für die Suche nach regulären
Ausdrücken";
$bool = ereg("[XYZxyz]", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.15: Suche nach Zeichengruppen

Ausgabe:

```
Nicht gefunden!
```

Das Suchmuster `[XYZxyz]`überprüft den String auf die letzten drei Buchstaben des Alphabets sowohl in Groß- als auch in Kleinschreibung. Da keiner der drei Buchstaben im String vorkommt, wird ein negatives Ergebnis ausgegeben. Dasselbe Ergebnis könnte auch mit `x|y|z|X|Y|Z` erzielt werden.

Um Tipparbeit zu vermeiden, können innerhalb von Zeichengruppen Bereiche definiert werden, die nicht mehr explizit angegeben werden müssen. Es wird einfach der Start- und Endpunkt festgelegt und durch ein Minuszeichen getrennt.

- `[A-Z]`: Alle Großbuchstaben zwischen A und Z
- `[a-z]`: Alle Kleinbuchstaben zwischen a und z
- `[0-9]`: Alle Ziffern

Diese Bereiche kann man nach Belieben zu Zeichengruppen zusammenfassen oder durch weitere Zeichen ergänzen.

```
<?
$str = "Die Zahl 2 ist kleiner als die Zahl 3 !";
$bool = ereg("[A-Za-z0-9! ]", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.16: Kombination von Zeichengruppen

Ausgabe:

```
Gefunden!
```

Das Beispielskript kombiniert die Zeichenbereiche aller Groß-/Kleinbuchstaben mit den Ziffern und dem Ausrufezeichen sowie dem Leerzeichen. Alle Suchmuster werden einfach aneinander gehängt und bilden so die gewünschte Zeichengruppe.

In der Praxis ist es oft einfacher, anstelle des Suchmusters den Bereich anzugeben, der nicht im String vorhanden sein soll. Für solche Fälle können Zeichengruppen negiert werden. Die geschieht einfach über das Karat-Zeichen am Anfang der Zeichengruppe.

```
<?
$str = "Die Zahl 2 ist kleiner als die Zahl 3 !";
$bool = ereg("[^A-Za-z0-9! ]", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.17: Negierung von Zeichengruppen

Ausgabe:

```
Nicht gefunden!
```

Maskierung von Sonderzeichen

Zur Steuerung der regulären Ausdrücke werden eine Handvoll Sonderzeichen genutzt, denen besondere Bedeutung zukommt. Dazu zählen unter anderem folgende Zeichen:

- `[ ]`
- `( )`
- `|`
- `\`
- `{ }`
- `+ -`

- ? .
- ^
- ?
- $

Wenn man nach einem dieser Zeichen suchen möchte, müssen sie durch einen Backslash markiert werden, damit der PHP Interpreter nicht falsch interpretiert. Dieses Vorgehen ist schon bekannt und funktioniert genauso wie zum Beispiel die Maskierung von doppelten Anführungszeichen in Strings.

Suchmuster	Ergebnis
\[\]	findet eckige Klammern
\\	findet Backslashs
\/\/	findet alle doppelten Slashs

Tabelle 8.4: Beispiele für maskierte Sonderzeichen

Arbeiten mit Wildcards

Der Einsatz von Wildcards (Platzhaltern) gehört zu den mächtigsten Instrumenten der regulären Ausdrücke. Sie erlauben es, flexible Suchmuster zu formulieren, mit denen Strings effektiv analysiert werden können. PHP unterstützt das gesamte Angebot der Wildcards innerhalb der regulären Ausdrücke, d.h. Sie können aus einer Reihe von Möglichkeiten wählen.

Innerhalb der regulären Ausdrücke unterscheidet man vier verschiedene Typen von Wildcards.

Wildcard	Beschreibung
.	steht für ein beliebiges Zeichen in einem String.
+	steht für beliebig viele Wiederholungen des vorhergehenden Zeichens.
*	steht für beliebig viele Wiederholungen des vorhergehenden Zeichens, kann aber auch ganz wegfallen.
?	das vorherstehende Zeichen ist optional.
.*	die Kombination der Wildcards Punkt und Sternchen schafft einen Platzhalter für eine beliebige Anzahl von Zeichen.

Tabelle 8.5: Wildcards für reguläre Ausdrücke

Das letzte Beispiel für die Kombination der Wildcards ist eine typische Arbeitsweise innerhalb der regulären Ausdrücke. Man setzt einfach die verschiedenen Möglichkeiten zusammen, bis das gewünschte Ergebnis eintritt.

In der Praxis sieht die Arbeit mit Wildcards so aus:

Beispiel	Beschreibung
`"Elef.nt"`	Findet Elefant, aber auch Elefint und Elefunt sowie jede andere mögliche Kombination.
`"Ele?fant"`	Findet Elefant, aber auch Elfant.
`"Ele+fant"`	Findet Elefant, aber auch Eleeefant und Eleeeeeefant. Aber nicht Elfant.
`"Ele*fant"`	Findet Elefant, aber auch Eleeefant und Eleeeeeefant. Im Gegensatz zu + wird auch Elfant gefunden.
`"Ele.*fant"`	Findet Elefant, sowie jede mögliche Buchstabenkombination innerhalb des Strings nach dem zweiten e.

Tabelle 8.6: Beispiele für Wildcards

Der reguläre Ausdruck mit den vermutlich größten Erfolgen wird nach diesem Kapitel eindeutig so aussehen:

```
$bool = ereg(".*", $str);
```

Egal welchen Wert $str hat, die Funktion `ereg()` würde immer TRUE zurückgeben. Über den Sinn des Ganzen lässt sich allerdings streiten.

Wiederholungen

Über geschweifte Klammern lässt sich innerhalb der regulären Ausdrücke festlegen, wie oft ein Zeichen sich wiederholen darf. Dabei kann genau festgelegt werden, in welchem Rahmen sich die Wiederholungen abspielen dürfen. Die Angaben innerhalb der geschweiften Klammern beziehen sich immer auf das vorherstehende Zeichen vor den Klammern.

```
$bool = ereg("Te{10}", $str);
```

Dieses Beispiel findet den String »Te« gefolgt von zehn weiteren »e«. Diese Angabe lässt keinen variablen Bereich zu. Möchte man eine flexible Anzahl von Wiederholungen festlegen, kann man weitere Angaben machen.

```
$bool = ereg("Te{3,10}", $str);
```

Dieses Beispiel findet alle Strings, die zwischen drei und zehn »e« anhängen haben. Für ein offenes Ende der Wiederholungen reicht es, wenn man die zweite Angabe einfach weglässt.

```
$bool = ereg("Te{3,}", $str);
```

Dieser Ausdruck findet jeden String, der mindestens drei »e« anhängen hat. Es gibt keine Begrenzung nach oben.

Wortanfang und Wortende

Über weitere Angaben innerhalb eines Suchstrings kann festgelegt werden, ob eine Suche nur am Anfang oder am Ende eines Strings durchgeführt werden soll. Dazu verwendet man innerhalb der regulären Ausdrücke das Karatzeichen und das Fragezeichen.

An dieser Stelle ist Vorsicht geboten, da das Karatzeichen auch für die Verneinung innerhalb von Zeichengruppen verwendet wird. Dies gilt allerdings nur innerhalb von eckigen Klammern. Steht das Zeichen außerhalb dieser Klammern, wird PHP angewiesen, nur am Anfang des Strings zu suchen.

Im Gegensatz dazu legt das Fragezeichen fest, dass nur am Ende des Strings gesucht werden soll.

Suchmuster	Beschreibung
»^Muster«	Sucht nur am Anfang des Strings.
»Muster?«	Sucht nur am Ende des Strings.

Tabelle 8.7: Einschränkung des Suche

```
<?
$str = "Der Weg zum Erfolg ist steinig";
$bool = ereg("^Erfolg", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.18: Suche am Anfang des Strings

Ausgabe:

```
Nicht gefunden!
```

Da das gesuchte Muster nicht am Anfang des Strings steht, wird keine Erfolgsmeldung ausgegeben. Wird allerdings am Ende des Strings nach »steinig« gesucht, dann werden wir fündig.

```
<?
$str = "Der Weg zum Erfolg ist steinig";
$bool = ereg("steinig$", $str);
if($bool) {echo "Gefunden!";} else {echo "Nicht gefunden!";}
?>
```

Listing 8.19: Suche am Ende des Strings

Ausgabe:

```
Gefunden!
```

Würde der String von einem Satzzeichen abgeschlossen werden, dann wäre die Suche erfolglos. Das Suchmuster muss exakt an der ersten oder letzten Stelle des Strings beginnen beziehungsweise enden.

Rangfolge und Klammerung

Die Reihenfolge innerhalb eines regulären Ausdrucks folgt einer bestimmten Festlegung, die eine eindeutige Zuordnung immer möglich macht. Diese Ordnung kann durch Klammersetzung verändert werden.

Stufe 1: Klammern `()`

Stufe 2: Gruppenoperatoren `+ * ? { }`

Stufe 3: Stringanfang und Stringende `^ $`

Stufe 4: Alternativen `|`

Innerhalb des regulären Ausdrucks können beliebig viele Klammern gesetzt werden, die es erlauben, die Rangfolge zu ändern. Dieser Eingriff ist allerdings nur bei wirklich komplexen Angaben nötig.

Das Beispiel aus der Einleitung

Erinnern Sie sich noch an das Beispiel am Anfang des Kapitels? Hier sind noch einmal die kompletten Angaben:

```
"Das erste Zeichen muss eine Zahl sein gefolgt von einem ‚e' oder
‚E'. Danach darf das Wort nur noch zehn Zeichen lang sein und muss
mit ‚.txt' enden."
```

Mit den Informationen der letzten Seiten ist der reguläre Ausdruck für diese Anfrage kein Problem mehr:

```
"[0-9]+e|E.{1,6}\.txt"
```

Das Beispielskript zeigt, dass es funktioniert:

```
<?
$liste = array("1Etext.txt","128EByte.txt",
               "7elese.txt","etextlab.txt",
               "134t.txt","32e.txt");

foreach ($liste as $var)
{
     $bool = ereg("[0-9]+e|E.{1,6}\.txt", $var);
     if($bool) {echo "Gefunden: $var<br>";}
}
?>
```

Listing 8.20: Reguläre Ausdrücke

Ausgabe:

```
Gefunden: 1Etext.txt
Gefunden: 128EByte.txt
Gefunden: 7elese.txt
Gefunden: 32e.txt
```

8.4.2 PHP-Funktionen für reguläre Ausdrücke

PHP unterstützt eine Handvoll Funktionen, die reguläre Ausdrücke verstehen und anwenden können. Die Syntax der regulären Ausdrücke kann ausschließlich in dieser Umgebung verwendet werden, da der Interpreter die Anweisungen sonst nicht versteht.

Die wichtigsten Funktionen sind in dieser Liste zusammengefasst:

Funktion	Beispiel	Erklärung
`ereg()`	`$bool = ereg($ra, $str);`	sucht nach einem Muster und gibt TRUE oder FALSE zurück
`eregi()`	`$bool = eregi($ra, $str, $liste);`	sucht nach einem Muster und gibt TRUE oder FALSE zurück; berücksichtigt keine Groß- und Kleinschreibung.
`ereg_replace()`	`$bool = ereg_replace($ra, $str2, $str);`	ersetzt Suchmuster in einem String

Tabelle 8.8: PHP-Funktionen für reguläre Ausdrücke

Funktion	Beispiel	Erklärung
eregi_replace()	$bool = eregi_replace($ra, $str2, $str);	ersetzt Suchmuster in einem String, ohne Groß- und Kleinschreibung
split()	$liste = split($ra, $str);	zerlegt einen String und gibt die Teile als Array zurück.
spliti()	$liste = spliti($ra, $str);	zerlegt einen String und gibt die Teile als Array zurück. Keine Groß- und Kleinschreibung

Tabelle 8.8: PHP-Funktionen für reguläre Ausdrücke (Forts.)

Die Funktion `ereg()` haben wir bereits zu Genüge kennen gelernt. Sie ermöglicht es, Strings mit regulären Ausdrücken auf bestimmte Suchmuster hin zu untersuchen. Die Funktion gibt TRUE oder FALSE zurück.

Die Funktion `eregi()` bewirkt dasselbe wie `ereg()`, mit dem Unterschied, dass keine Groß- und Kleinschreibung berücksichtigt wird. Im Detail bedeutet dies, dass der Ausdruck »Hand« auch den String »hand« findet.

Interessant wird es bei der Funktion `ereg_replace()`, die, wie der Name schon sagt, für das Suchen und Ersetzen von Strings zuständig ist. Alle Strings, die mit dem regulären Ausdruck übereinstimmen, werden durch den Ersatzstring ausgetauscht.

```
<?
$str = "Hand Kind Lied Land Leid Lob Grab Hund Saat";
$newstr = ereg_replace("[LH].nd", "ERSETZT" ,$str);
echo $newstr;
?>
```

Listing 8.21: Suchen und Ersetzen mit regulären Ausdrücken

Ausgabe:

```
ERSETZT Kind Lied ERSETZT Leid Lob Grab ERSETZT Saat
```

Die Funktion arbeitet genau wie die PHP-Funktion `str_replace()`, die ebenfalls Strings sucht und ersetzt. Allerdings bietet `ereg_replace()` bedeutend mehr Möglichkeiten und ist wesentlich flexibler. Diese Freiheit wird allerdings durch eine langsamere Verarbeitung erkauft.

Genau wie `eregi()` ignoriert die Funktion `eregi_replace()` Groß- und Kleinschreibung im String. Dieses Feature macht den Umgang mit manchen Problemen leichter.

Das letzte Funktionspärchen in dieser Reihe ist `split()` und `spliti()`. Diese Funktionen erlauben es, Strings aufzuspalten und die Daten in Arrays zu speichern. Genau wie bei der Funktion `explode()` muss ein Suchmuster angegeben werden, an dem der String gespalten werden soll.

```
<?
$str = "Ich habe heute ein Haus, einen Hund, eine Katze
und einen Bären gesehen.";
$liste = split(" |, |\.", $str);
foreach ($liste as $var)
{
    echo "$var<br>";
}
?>
```

Listing 8.22: split() zerlegt einen Satz

Ausgabe:

```
Ich
habe
heute
ein
Haus
einen
Hund
eine
Katze
und
einen
Bären
gesehen
```

Mit dieser einfachen Anwendung ist es möglich, einen Satz in seine Bestandteile zu zerlegen, ohne dabei auf Satzzeichen Rücksicht nehmen zu müssen. Dank des regulären Ausdrucks werden die Satzzeichen einfach entfernt.

Die Funktion `spliti()` leistet dasselbe wie `split()`, ohne dabei die Groß- und Kleinschreibung im String zu beachten.

8.4.3 Übungen

Ü

1. Was sind reguläre Ausdrücke?
2. Was ist der Unterschied zwischen Zeichen und Zeichengruppen?
3. Erklären Sie den Unterschied zwischen den regulären Ausdrücken A|B|C und [ABC] .

4. Welchen Bereich deckt die folgende Zeichengruppe ab: [A-No-z] ?
5. Was sind Wildcards?
6. Warum gibt das folgende Skript FALSE aus?

```
<?
  $str = "PHP ist toll!";
  $bool = ereg("PHP$", $str);
  echo $bool;
  ?>
```

7. Was ist der Unterschied zwischen `ereg()` und `eregi()`?

8.5 Umgebungsvariablen

Wenn ein HTTP-Client eine Anfrage an einen Webserver stellt, werden innerhalb dieser Kommunikation eine Reihe von Informationen ausgetauscht, die für den Datentransfer nötig sind. Unter anderem werden Informationen wie die IP-Adresse oder der Protokollname übergeben, damit beide Seiten wissen, auf welcher Ebene sie kommunizieren müssen und vor allem wohin die Daten geschickt werden sollen.

Wie auch alle anderen Websprachen unterstützt PHP eine Funktion, die es erlaubt, Umgebungsvariablen des Servers auszulesen. Jeder Wert steht während der Laufzeit des Skriptes über eine Schlüsselkonstante zur Verfügung, über die der Wert abgerufen werden kann.

Die Funktion ist sehr einfach und hört auf den schönen Namen `getenv()`:

```
$str = getenv(NAME);
```

Als Parameter wird der Name der Umgebungsvariablen in Anführungszeichen übergeben. Die Funktion gibt den Wert der Umgebungsvariablen als String zurück. Das folgende Skript gibt beispielsweise die IP-Adresse des anfragenden Clientrechners aus.

```
<?
$ip = getenv("REMOTE_ADDR");
echo "Ihre IP-Adresse ist: $ip";
?>
```

Listing 8.23: Die IP-Adresse wird ausgegeben

Ausgabe:

```
192.168.100.12
```

Die IP-Adresse identifiziert jeden Rechner eindeutig im Internet auf der ganzen Welt. Eine Austausch von Daten kann nur erfolgen, wenn die IP-Adressen gegenseitig bekannt sind. In den meisten Fällen bekommt man diese Folge von Zahlen automatisch von seinem Provider zugewiesen. Jedes mal, wenn Sie sich neu einwählen, wird Ihnen eine neue Adresse zugeteilt.

Neben der IP-Adresse stehen noch eine ganze Reihe weiterer Informationen zur Verfügung. Diese Liste verschafft einen Überblick über die wichtigsten Umgebungsvariablen.

Name	Beschreibung
REMOTE_ADDR	IP-Adresse des anfragenden Rechners
REMOTE_HOST	Hostname des anfragenden Rechners
REMOTE_USER	Username des Rechners, falls bekannt
SERVER_SOFTWARE	Bezeichnung der Serversoftware
SERVER_NAME	Hostname des Servers oder IP-Adresse
SERVER_PROTOCOL	Protokoll der Anfrage
SERVER_PORT	Port der Anfrage
REQUEST_METHOD	Anfrage Methode (post / get)
SCRIPT_NAME	Name des ausführenden Skriptes
QUERY_STRING	Übergabedaten in der URL einer GET-Anfrage
AUTH_TYPE	Methode der Userüberprüfung
CONTENT_TYPE	Übergabedaten einer POST-Anfrage
CONTENT_LENGTH	Anzahl der Zeichen in der POST-Anfrage
HTTP_USER_AGENT	Bezeichnung des anfragenden Browsers
HTTP_ACCEPT	MIME-Typen, die vom Browser akzeptiert werden

Tabelle 8.9: Umgebungsvariablen

Die Umgebungsvariablen REMOTE_USER und AUTH_TYPE werden vom Server nur gesetzt, wenn die entsprechenden Informationen vorliegen. Das ist in der Regel dann der Fall, wenn Webseiten einer Zugriffsbeschränkung unterliegen. In den meisten Fällen geschieht das über einen htaccess-Passwortschutz, der vom Surfer einen Usernamen und ein Passwort verlangt. Sind die Daten korrekt, dann wird die Seite freigegeben.

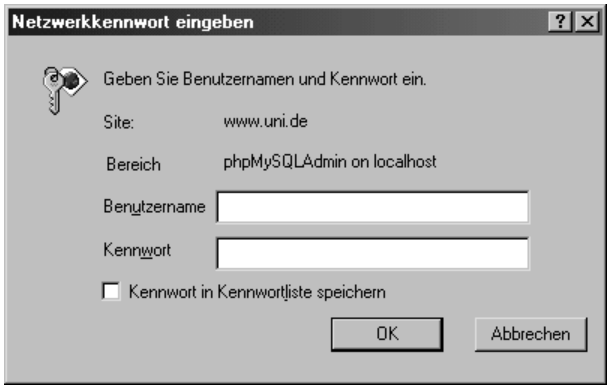


Abbildung 8.6: htaccess-Passwortschutz

Der Server speichert den Usernamen in der Umgebungsvariablen REMOTE_USER und kann über `getenv()` ausgelesen werden. So ist es möglich, eine PHP-Seite an ein bestimmtes Userprofil anzupassen.

Das folgende Programm zeigt alle hier besprochenen Umgebungsvariablen in der Praxis. Sie werden ausgelesen und anschließend wieder im Browser ausgegeben.

```
<?
$data=getenv("REMOTE_ADDR");
echo "$data <br>";
$data=getenv("SERVER_SOFTWARE");
echo "$data <br>";
$data=getenv("SERVER_NAME");
echo "$data <br>";
$data=getenv("SERVER_PROTOCOL");
echo "$data <br>";
$data=getenv("SERVER_PORT");
echo "$data <br>";
$data=getenv("REQUEST_METHOD");
echo "$data <br>";
$data=getenv("SCRIPT_NAME");
echo "$data <br>";
$data=getenv("HTTP_USER_AGENT");
echo "$data <br>";
$data=getenv("HTTP_ACCEPT");
echo "$data <br>";
?>
```

Listing 8.24: Alle Umgebungsvariablen auf einen Blick

Ausgabe:

```
127.0.0.1
Apache/1.3.12 (Win32)
master
HTTP/1.1
80
GET
/php4/php.exe
Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)
image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/
msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, */*
```

Umgebungsvariablen sind ein einfaches und gleichzeitig sehr mächtiges Werkzeug, um mehr über einen User zu erfahren. Die Daten, die ein Surfer ins Web schickt, können eine Menge über ihn aussagen.

Ein praktisches Beispiel ist die Abfrage des Browsertyps. Mit dieser Information kann der Programmierer eventuelle Besonderheiten in der HTML-Gestaltung berücksichtigen oder ein JavaScript für einen Browser anpassen.

Eine weitere wichtige Information ist die IP-Adresse dieses Surfers. In vielen sicherheitsrelevanten Situationen kann es sehr nützlich sein, die IP-Adresse des Surfers zu haben, um auf seine Identität schließen zu können. Die Webseite des RIPE Network Coordination Centre unter *www.ripe.net* kann hier hilfreich sein.

9 Objektorientierte Programmierung

PHP unterstützt das Konzept der objektorientierten Programmierung (OOP), unterwirft sich aber nicht der strengen Reglementierung, die das Prinzip der OOP nach sich zieht. Der Programmierer hat fast in jeder Situation die Wahl, ob er ein Programm oder einen Algorithmus objektorientiert programmieren will oder nicht.

PHP ist keine objektorientierte Sprache im strengen Sinne. Einige Funktionalitäten innerhalb der Sprache werden zwar mit objektorientierten Ansätzen gelöst, sind aber auch ohne das Hintergrundwissen über die OOP verständlich. Im Allgemeinen kann man sagen, dass Skriptsprachen es immer schwerer haben, das Konzept der Objektorientierten Programmierung umzusetzen, da es ihnen meistens an Grundvoraussetzungen mangelt. Angefangen bei der automatisierten Typisierung von Werten bis hin zum einfachen Aufbau eines Programms gibt es einige Punkte, die der OOP im Wege stehen.

Trotz allem ist es möglich, ein PHP-Skript objektorientiert aufzubauen. Dazu müssen allerdings erst einige Grundlagen geklärt werden.

9.1 Kapitelübersicht

- Was ist die objektorientierte Programmierung?
- Klassen und Objekte
- Vererbung
- Wichtige Begriffe der OOP
- OOP mit PHP
- Informationen über Klassen und Objekte

9.2 Was ist objektorientierte Programmierung?

Eine der größten Stärken des Menschen ist die Fähigkeit, Lebewesen und Dinge seiner Umwelt in bestimmte Muster einzuordnen und zu typisieren. Alle Erfahrungen werden in bestimmte Schubladen gesteckt und mit

Merkmalen und Eigenschaften versehen, die es erlauben, sie wiederzuerkennen und mit anderen Erfahrungen in Beziehungen zu setzen.

Ein Hund auf der Straße wird immer als ein Hund erkannt werden, aber gleichzeitig wird dieser spezielle Hund auch in die Schublade der Tiere eingeordnet und darüber hinaus in die Gruppe der Lebewesen. Diese Reihe kann man immer weiterspinnen, bis man zu einem obersten gemeinsamen Punkt kommt, der alle Erfahrungen, Eindrücke und Dinge im Gedächtnis vereint. Es bildet sich eine Art Assoziationskette, die alle Begriffe innerhalb eines Netzwerks definiert.

Diese zugegebenermaßen sehr allegorische Vorrede soll Ihnen einen kurzen Einblick in die Denkweise der objektorientierten Theorie geben. Es ist immer sehr schwierig, die Ansätze der OOP in einem einzigen Kapitel erläutern zu wollen, ohne dabei auf die wesentlichen Punkte zu verzichten. Meiner Meinung nach ist der beste Einstieg immer über ein Beispiel.

Theoretisch gesehen besteht ein Hund aus einer Ansammlung von Eigenschaften und Fähigkeiten, die ihn genau als das Lebewesen identifizieren, das er ist. Ein schnurrender Hund würde genauso aus dem Rahmen fallen wie eine bellende Katze.

Der Hund als Objekt ist also nur dann sinnvoll, wenn alle Eigenschaften und Fähigkeiten zusammengefasst werden können, damit sie als Einheit auftreten. Es gibt keine Eigenschaft Fellfarbe oder die Fähigkeit bellen, die für sich allein im Raum steht, sondern nur einen Verbund von Merkmalen, die miteinander kommunizieren.

Die objektorientierte Programmierung definiert die Zusammenfassung von Eigenschaften und Fähigkeiten als ein Objekt. Der Vorgang der Zusammenfassung wird als Kapselung bezeichnet. In einer Programmiersprache werden die Fähigkeiten eines Objekts über Funktionen realisiert, die als Objektmethoden bezeichnet werden. Die Eigenschaften eines Objekts werden in Objektvariablen definiert. Ein streng objektorientiertes System besteht nur als Objekten mit den entsprechenden Funktionen und Variablen. Globale Werte sind nicht erlaubt.

9.2.1 Objekte und Klassen

Objekte dienen dazu, die Realität zu abstrahieren und in einem Modell darstellen zu können. Das Objekt Hund könnte zum Beispiel in einer sehr einfachen Form aus folgenden Methoden und Variablen bestehen:

- Methode bellen
- Methode wedeln
- Variable Fellfarbe
- Variable Alter
- Variable Größe

Jeder Hund beziehungsweise jedes Objekt in einem Programm definiert sich über diese Vorgaben und wird so als das Objekt Hund erkannt. Allerdings ist kein Hund wie der andere: Ein Blick auf die Straßen und Parks der Welt genügt um zu sehen, dass es tausende von Hunden gibt, von denen aber keiner dem anderen gleicht. Jedes Objekt speichert andere Werte und unterscheidet sich so von allen anderen Objekten.

Die Gemeinsamkeiten bleiben aber unverkennbar: Alle Objekte haben den gleichen Aufbau und die gleiche Struktur. Die Objekte unterscheiden sich zwar phenotypisch voneinander, sind aber genotypisch vom gleichen Typ. Diese Gemeinsamkeiten erlauben es, alle Objekte vom gleichen Typus in so genannten Klassen zusammenzufassen, die als eine Art Bauplan für ein Objekt fungieren.

Es entsteht ein grundsätzliches Konzept für ein Objekt, das genau festlegt, wie der Aufbau und die Struktur dieses Typs auszusehen haben. Programmiertechnisch gesehen wird durch eine Klasse ein neuer Datentyp erschaffen, der ähnlich wie Integer oder Gleitpunktzahlen bestimmte Werte mit ihren Eigenschaften definiert. Eine Klasse ist ein theoretisches Modell der Realität, das erst durch Objekte realisiert wird.

Immer wenn ein neues Objekt erschaffen wird, muss auf die Informationen der Klasse zugegriffen werden. Danach existiert es autark im Programm und ist unabhängig von der Klasse. Wird die Klasse in ihrer Struktur allerdings geändert, dann wirkt sich das auf alle Objekte dieser Klasse aus.

9.2.2 Vererbung

Ein wesentlicher Punkt in der objektorientierten Programmierung ist die Vererbung. Diese Technik beschreibt den Vorgang, wenn eine neue Klasse erschaffen wird, die auf der Struktur einer schon bestehenden Klasse beruht. Die neue Klasse erbt dabei alle Merkmale und Methoden der alten Klasse und erweitert dieses Repertoire durch eigene Merkmale.

Beispielhaft könnte man sich diesen Vorgang anhand der Klasse Dackel vorstellen, die aus der Klasse Hund abgeleitet wird. Ein Dackel ist zweifellos ein Hund, aber er unterscheidet sich durch eine Reihe von Merkmalen von den meisten anderen Hunden. Die Klasse Dackel erbt alle Eigenschaften der Klasse Hund und erweitert diese durch eigene Merkmale.

Die Mutterklasse einer solchen Beziehung nennt man Superklasse, die erbende Klasse wird als Subklasse bezeichnet. Es ist durchaus möglich, dass eine Subklasse wiederum eine neue Klasse beerbt und so zu einer Superklasse wird. In komplexen objektorientierten Programmen ist es die Regel, dass alle Klassen durch ein Netzwerk von Superklassen und Subklassen miteinander verbunden sind, so dass alle Eigenschaften und Methoden voneinander abhängen.

Generell sind Subklassen unabhängig von ihren Superklassen. Allerdings würde eine Änderung an einer Superklasse alle abhängigen Subklassen ebenfalls betreffen.

Ein Streitpunkt in der objektorientierten Programmierung ist das Schlagwort »Mehrfachvererbung«. Diese Technik beschreibt die Möglichkeit, eine Klasse aus mehreren Superklassen zu erstellen. Die neue Klasse würde in einem solchen Fall sämtliche Eigenschaften aller Superklassen erben. Einige Punkte sprechen gegen den Einsatz der Mehrfachvererbung, wie zum Beispiel eine chaotische Netzwerkstruktur oder undurchsichtige Klassenzusammenhänge.

Methoden überscheiben

Ein wichtiger Punkt im Zusammenhang mit der Vererbung ist die Möglichkeit der Neudefinition von Methoden einer Klasse. Wenn eine Klasse durch Vererbung ihre gesamte Funktionalität an eine neue Klasse weitergibt, erbt diese Klasse auch alle Methoden. In manchen Situationen ist es möglich, dass die geerbte Methode verändert werden muss, um den Ansprüchen dieser Klasse gerecht zu werden. Der Programmierer hat also die Möglichkeit eine bereits bestehende Methode zu verändern.

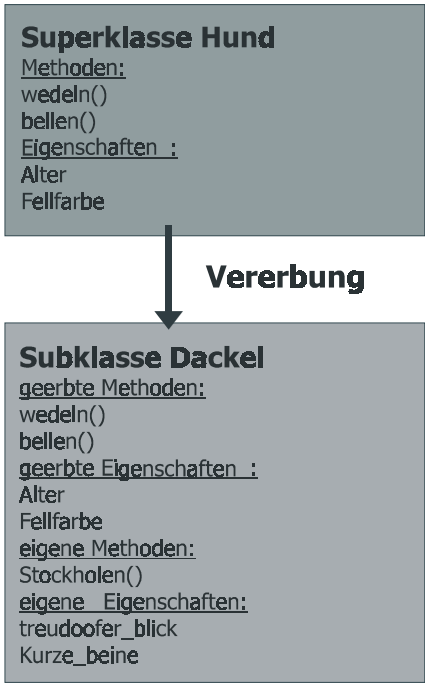


Abbildung 9.1: Vererbung

Diese Technik wird Überschreiben genannt, da die bestehende Methode der Superklasse durch die neue Methode überschrieben wird. Die neue Methode ist nur innerhalb der neuen Klasse gültig, die alte Klasse wird von diesem Vorgang nicht berührt.

9.2.3 Wichtige Begriffe der OOP

- Eine **Klasse** ist eine Strukturvorlage für ein Objekt. Sie besteht aus Methoden (Funktionen) und Variablen, die den Wert und das Verhalten einer Klasse definieren. Klassen können ihre Charakteristik an Subklassen vererben.
- Ein **Objekt** ist die Instanz einer Klasse. Es beschreibt eine mögliche Ausprägung einer Klassenstruktur.
- Eine **Objektmethode** dient einer Klasse als Interaktionsmöglichkeit mit der Außenwelt. Die Gesamtheit der Methoden einer Klasse bestimmt ihr Verhalten im Programm. Eine Methode wird als Funktion realisiert.

- Eine **Objektvariable** speichert den Wert eines Attributes in einem Objekt.
- Eine **Superklasse** vererbt ihre Methoden und Variablen an eine neue Klasse. Sie befindet sich im Klassennetzwerk über einer Subklasse.
- Eine **Subklasse** erbt alle Methoden und Variablen einer Superklasse. Sie steht im Klassennetzwerk unter einer Superklasse.
- Die Technik der **Vererbung** beschreibt den Vorgang, wenn eine neue Klasse erschaffen wird, die auf der Struktur einer schon bestehenden Klasse aufbaut. Die neue Klasse erbt dabei alle Merkmale und Methoden der alten Klasse und erweitert dieses Repertoire durch eigene Merkmale.
- Das **Überschreiben** von Methoden ist immer dann nötig, wenn eine geerbte Methode neu definiert werden muss. Dabei wird dieser Methode eine neue Funktionalität zugewiesen.

9.3 OOP mit PHP

Da PHP von der Konzeption her keine streng objektorientierte Sprache ist, steht es dem Programmierer frei, diese Systematik zu nutzen. Es ist durchaus möglich, einen Teil des Programms objektorientiert zu realisieren, während ein anderer Teil in »normaler« Syntax verfasst wird. Meistens ist es eine Frage der inneren Logik, ob ein Programm objektorientierte Teile enthalten soll oder nicht.

Um die objektorientierte Theorie aus dem letzten Kapitel in die Praxis umzusetzen, bietet es sich an, das Hunde-Dackel-Beispiel Schritt für Schritt in PHP zu realisieren. Als erster Schritt sollte also eine Klasse mit dem Namen Hund geschaffen werden.

9.3.1 Klassen in PHP

Eine Klassendefinition in PHP beginnt immer mit dem Schlüsselwort `class`. Damit wird dem Interpreter signalisiert, dass das folgende Programmkonstrukt eine Klasse ist und nicht ausgeführt werden soll. Nach dem Schlüsselwort `class` folgt der Name der Klasse, der denselben Namenskonventionen unterworfen ist wie eine Variable. Dann wird in geschweiften Klammern die eigentliche Klassenstruktur definiert.

```
class Name
{ ... }
```

Innerhalb der geschweiften Klammern werden die Eigenschaften und das Verhalten der Klasse definiert. Grundlage für diese Informationen sind die Variablen und Methoden der Klasse, die in der Klassendeklaration angegeben werden.

In PHP werden Methoden als einfache Funktionen dargestellt und darum über das Schlüsselwort `function` angegeben. Die Klassenvariablen werden ähnlich einfach gehandhabt. Die Syntax unterscheidet sich nicht von den bisher verwendeten Variablen.

```
class Name
{
function func1()
   {...}
   var $x;
   var $y;
}
```

Jede Methode einer Klasse steht für ein bestimmtes Verhalten oder für eine bestimmte Interaktion mit der Umwelt. Die Variablen der Klasse speichern Informationen und Eigenschaften über das zukünftige Objekt.

Der komplette Code für die Klasse Hund würde also so aussehen:

```
<?
class Hund
{
        function wedeln()
        {
                echo "Der Hund wedelt!<br>";
        }
        function bellen()
        {
                echo "WauWauWau!<br>";
        }
        var $Alter;
        var $Fellfarbe;
}
?>
```

Listing 9.1: Die Klasse Hund

Die Klasse Hund definiert das komplette Verhalten und alle Eigenschaften unseres zugegebenermaßen sehr rudimentären Hundes. Ansonsten macht das Programm nichts! Wenn Sie das PHP-Skript ausführen, dann werden Sie eine leere Browserseite zu sehen bekommen. Um diese Klasse zum Leben zu erwecken, brauchen wir ein Objekt.

9.3.2 Objekte in PHP

Um eine Klasse nutzen zu können, muss daraus ein Objekt gezogen werden. Ein neues Objekt wird immer über das Schlüsselwort *new* instanziiert und wird wie eine Variable behandelt.

```
$objekt = new klassenname();
```

Hinter dem Schlüsselwort `new` steht der Name der betroffenen Klasse, aus der das Objekt gezogen werden soll. Die abschließenden Klammern dienen dazu, eventuelle Parameter an einen Konstruktor zu übergeben. Hierzu kommen wir in einem späteren Kapitel.

Wenn ein Objekt erfolgreich erstellt wurde, dann kann über den Namen des Objektes auf die komplette Funktionalität der Klasse zugegriffen werden. PHP stellt zu diesem Zweck den Operator `->` zur Verfügung, der das Objekt mit seinen Methoden und Variablen verknüpft. Um beispielsweise eine Objektmethode aufzurufen, muss man folgendermaßen vorgehen:

```
$objekt -> methode();
```

So ruft der PHP-Interpreter die Methode des Objekts auf, wie sie in der Klasse definiert wurde. Um eine Variable auszulesen oder zu setzen, geht man denselben Weg:

```
$objekt -> variable = 100;
echo $objekt -> variable;
```

Objektvariablen können sowohl gesetzt als auch gelesen werden. Es ist wichtig, dass beim Zugriff auf Objektvariablen kein Dollarzeichen gesetzt wird.

Mit diesen Voraussetzungen kann die Klasse Hund nun zum Leben erweckt werden. Der besseren Übersicht wegen wird die Klassendefinition einfach per `include()` in das Programm eingelesen.

```
<?
include ("listing108.php");
$bello = new hund();
$bello -> fellfarbe = "Braun";
$bello -> alter = 5;
$bello -> bellen();
$bello -> wedeln();
echo "Fellfarbe: ". $bello -> fellfarbe;
echo "<br>";
echo "Alter: ". $bello -> alter;
?>
```

Listing 9.2: Das Objekt $bello

Ausgabe:

```
WauWauWau!
Der Hund wedelt!
Fellfarbe: Braun
Alter: 5
```

Wie die Ausgabe beweist, existiert während der Laufzeit des Programms ein Objekt mit dem Namen `$bello`. Dieses Objekt der Klasse Hund zeichnet sich durch ein braunes Fell aus und ist im besten Alter von 5 Jahren. Bello kann sowohl bellen als auch wedeln: ein ganzer Kerl!

9.3.3 Vererbung in PHP

Der nächste logische Schritt ist jetzt ein Spielgefährte für Bello in Form eines Dackels. Da beide Objekte nach wie vor der Rasse Hund angehören, bietet es sich an, die Klasse Dackel aus der Klasse Hund abzuleiten und so alle wesentlichen Merkmale zu erben.

Über das Schlüsselwort `extends` (in Anlehnung an JAVA) kann aus einer bestehenden Klasse eine neue Klasse erschaffen werden. Die neue Klasse erbt dabei automatisch alle Methoden und Variablen der Superklasse und kann gleichzeitig die bestehende Struktur nach Belieben erweitern.

```
class Klasse1
{...}
class Klasse2 extends Klasse1
{...}
```

Wesentlich ist natürlich, dass die potenzielle Superklasse schon existiert, sonst kann nämlich nichts vererbt werden.

In unserem Beispielprogramm soll die Klasse Dackel aus der Klasse Hund abgeleitet werden. Dazu bedienen wir uns wieder der schon bestehenden Klassendefinition aus den Listing 108.

```
<?
include ("listing108.php");
class Dackel extends Hund
{
        function stockholen()
        {
                echo "Der Dackel holt den Stock!<br>";
        }

        var $treudoofer_Blick = TRUE;
        var $kurze_Beine = TRUE;
}
$waldi = new dackel();
$waldi -> alter = 7;
$waldi -> bellen();
$waldi -> wedeln();
$waldi -> stockholen();
echo $waldi -> alter . "<br>";
if ($waldi -> kurze_Beine) {echo "Der Dackel ertrinkt
fast in einer Pfütze!<br>";}
?>
```

Listing 9.3: Ein Dackel wird geboren …

Ausgabe:

```
WauWauWau!
Der Hund wedelt!
Der Dackel holt den Stock!
7
Der Dackel ertrinkt fast in einer Pfütze!
```

Das Skript zeigt, dass das Objekt `$waldi` sowohl Methoden und Variablen der Klasse Hund als auch der Klasse Dackel benutzen kann. Diese »Freiheit« verdankt Waldi dem Erbe aus der Superklasse Hund, da er alle Methoden und Eigenschaften übernommen hat. Um die Dackelcharakteristik vom einfachen Hund abzuheben, wurden lediglich die dackeltypischen Merkmale hinzugefügt.

Methoden überschreiben in PHP

Um der Klasse Dackel nun den letzten Schliff zu geben, muss die Methode `bellen()` angepasst werden. Jedes Dackelobjekt soll sofort am Klang des

Bellens von einem normalen Hundeobjekt zu unterscheiden sein. Um das zu erreichen muss die Methode `bellen()` der Klassen Hund überschrieben werden, um sie mit einer eigenen Funktionalität zu versehen.

```
<?
include ("listing108.php");
class Dackel extends Hund
{
        function bellen()
        {
                echo "KläffKläffKläff!<br>";
        }

        function stockholen()
        {
                echo "Der Dackel holt den Stock!<br>";
        }

        var $treudoofer_Blick = TRUE;
        var $kurze_Beine = TRUE;
}
$bello = new Hund();
$waldi = new Dackel();
$bello -> bellen();
$waldi -> bellen();
?>
```

Listing 9.4: Überschreiben von Methoden

Ausgabe:

```
WauWauWau!
KläffKläffKläff!
```

Die Ausgabe des Skripts beweist, dass die Methode `bellen()` erfolgreich überschrieben wurde. Das Dackelobjekt $waldi hat nun eine ganz eigene Klangfärbung erhalten und ist leicht vom Hund $bello zu unterscheiden.

9.3.4 Klassen und der Konstruktor

PHP unterstützt den Einsatz von Konstruktoren in der objektorientierten Programmierung. Der Konstruktor einer Klasse ist eine Methode, die den gleichen Namen trägt wie die Klasse selbst. Sie hat die implizite Eigenschaft, automatisch bei der Objekterstellung aufgerufen zu werden.

```
<?
class Dackel
{
        function Dackel()
        {
                echo "Ein Dackel wird geboren!";
        }
}
$waldi = new Dackel();
?>
```

Listing 9.5: Der Dackelkonstruktor

Ausgabe:

```
Ein Dackel wird geboren!
```

Die Konstruktormethode kann dazu verwendet werden, bestimmten Objektvariablen Werte zuzuweisen oder weitere Methoden, die für den Aufbau des Objekts nötig sind, zu starten. Im Gegensatz zu anderen objektorientierten Sprachen werden die Konstruktoren von Superklassen nicht automatisch mitaufgerufen.

Konstruktor mit Parametern

Technisch gesehen handelt es sich bei dem Konstruktor um eine einfache Methode, die vom PHP-Interpreter automatisch aufgerufen wird. Wie jede Methode kann der Konstruktor einer Klasse einen oder mehrere Parameter übernehmen und weiter verarbeiten. Das Vorgehen unterscheidet sich nicht von einer normalen Funktion.

Die Werte für die Übergabe werden bei der Objekterstellung in Klammern angegeben. Sie stehen dann in der Methode zur Verfügung:

```
<?
class Dackel
{
        function Dackel($name, $Alter)
        {
        echo "Der Dackel heißt $name und ist $Alter Jahre
alt!";
        }
}
$waldi = new Dackel("Waldi", 8);
?>
```

Listing 9.6: Ein Konstruktor mit Parametern

Ausgabe:

```
Der Dackel heißt Waldi und ist 8 Jahre alt!
```

Wie bei einer normalen Funktion übergibt der Konstruktor die Parameter an die Methode weiter. Im Beispiel werden sie einfach über einen String ausgegeben, der Informationen über das aktuelle Objekt enthält.

Die aktuelle Version von PHP4 unterstützt nicht die Möglichkeiten des Destruktors. Dieser wird automatisch aufgerufen, wenn ein Objekt wieder zerstört wird. Das ist in der Regel dann der Fall, wenn das Programm endet.

9.3.5 Zugriffsmethoden mit $this

Die Systemvariable `$this` wird automatisch vom PHP-Interpreter initialisiert, wenn ein Objekt erstellt wird. Sie enthält einen internen Zeiger auf das aktuelle Objekt und wird immer dann eingesetzt, wenn innerhalb dieser Instanz auf einen Wert zugegriffen werden muss.

Das gängigste Beispiel für den Einsatz von `$this` sind Zugriffsmethoden für die Werte eines Objekts. Zugriffsmethoden dienen zum Lesen und Setzen von Objektvariablen und sollen den Umgang mit komplexen Strukturen erleichtern.

```
<?
class Dackel
{
        function Dackel($str)
        {
                $this->name=$str;
                echo "Name: ". $this->name;
        }
        function setAlter($num)
        {
                $this->alter=$num;
        }
        function getAlter()
        {
                return $this->alter;
        }
        var $alter;
        var $name;
}
$waldi = new Dackel("Waldi");
```

```
$waldi->setalter(2);
echo "<br>Alter: ". $waldi->getalter();
?>
```

Listing 9.7: Die Variable $this mit Zugriffsmethoden

Ausgabe:

```
Name: Waldi
Alter: 2
```

Der erste Einsatz von `$this` ist bereits ganz am Anfang des Skripts im Konstruktor zu sehen. Die Variable `$name` wird automatisch auf den Wert des Übergabestrings gesetzt, der in der nächsten Zeile zur Kontrolle sofort wieder ausgegeben wird. Zusätzlich existieren zwei Zugriffsmethoden für die Variable `$alter`, über die der Wert gesteuert werden kann. Beide Methoden arbeiten ebenfalls mit der Variable `$this`.

Die gesamte Funktion des Skripts beschränkt sich auf die Ausgabe sämtlicher Werte.

9.3.6 Methoden aus Klassen aufrufen

Seit der Version 4 unterstützt PHP die Möglichkeit aus einem Objekt heraus die Methoden anderer Klassen aufzurufen. Dieses Vorgehen ist zwar nicht wirklich korrekt objektorientiert, kann aber in manchen Situationen nützlich sein.

Die Syntax ist denkbar einfach: Um eine Methode einer fremden Klasse aufzurufen, reicht es, die Zielklasse mit zwei Doppelpunkten vor den Funktionsaufruf zu stellen.

```
Klasse:: Methode();
```

Der Rest wird von PHP erledigt. In der Praxis sieht die Sache dann so aus:

```
<?
class Hund
{
        function hundebellen()
        {
                echo "WauWauWau!";
        }
}
class Dackel
{
```

```
        function dackelbellen()
        {
                Hund::hundebellen();
        }
}
$waldi = new Dackel();
$waldi -> dackelbellen();
?>
```

Listing 9.8: Bellende Dackel

Ausgabe:

```
WauWauWau!
```

Im Beispielskript leiht sich das Dackelobjekt die Stimme der Klasse Hund um sich Gehör zu verschaffen. In der Methode `dackelbellen()` wird die Methode `hundebellen()` aus der Klasse Hund aufgerufen. Es ist nicht mal nötig, dass ein Objekt der Klasse Hund existiert, um diese Technik einzusetzen.

9.3.7 Übungen

1. Was unterscheidet die objektorientierte Programmierung von anderen Theorien?
2. Was ist der Unterschied zwischen einer Methode und einer Funktion?
3. Was wird zwischen Klassen vererbt?
4. Was unterscheidet Objekte von Klassen?
5. Welche Funktion hat das Schlüsselwort extends?
6. Welchen Sinn macht ein Konstruktor?

9.4 Informationen über Klassen und Objekte

Genau wie bei primitiven Datentypen stellt PHP eine Reihe von Werkzeugen zur Verfügung, mit denen der Programmierer Informationen über Klassen und Objekte abfragen kann. Zu den verfügbaren Informationen zählen die Namen von Klassen und Objekten sowie Angaben über deren Struktur.

Funktion	Parameter	Beschreibung
`get_class()`	Objekt	gibt die Klasse eines Objekts zurück
`get_parent_class()`	Objekt	gibt die Superklasse eines Objekts zurück
`get_object_vars()`	Objekt	gibt ein Array mit allen Objektvariablen zurück
`get_class_methods()`	Klasse	gibt ein Array mit allen Methoden einer Klasse zurück
`is_subclass_of()`	Objekt, Klasse	bestimmt, ob ein Objekt zu einer Subklasse der angegebenen Klasse gehört
`method_exists()`	Objekt	überprüft, ob eine Methode in einem Objekt existiert
`class_exists()`	Klasse	überprüft, ob eine Klasse existert

Tabelle 9.1: Informationen über Klassen und Objekte

```
<?
include("listing108.php");
class Dackel extends Hund
{
        function bellen()
        {
                echo "KläffKläffKläff!<br>";
        }

        function stockholen()
        {
                echo "Der Dackel holt den Stock!<br>";
        }

        var $treudoofer_Blick = TRUE;
        var $kurze_Beine = TRUE;
}
$bello = new Hund();
$fiffi = new Dackel();

echo get_class($bello) ."<br>";
echo get_class($fiffi) ."<br>";
echo get_parent_class($fiffi) ."<br>";
$liste = get_object_vars($fiffi);
echo count($liste) ."<br>";
if(is_subclass_of($fiffi, "Hund")) {$liste =
get_class_methods("Dackel");}
```

```
echo $liste[0]," ", $liste[1], "<br>";
if (method_exists($bello, "bellen")) {echo "Die Methode bellen steht
im Objekt \$bello zur Verfügung!<br>";}
if (class_exists("Koeter")) {echo "Die Klasse Koeter steht zur Vefü-
gung!";}
?>
```

Listing 9.9: Kontrollfunktionen für Objekte und Klassen

Ausgabe:

```
hund
dackel
hund
2
bellen stockholen
Die Methode bellen steht im Objekt $bello zur Verfügung!
```

Das Skript zeigt, wie die Klassen Hund und Dackel mit ihren zugehörigen Objekten auf Herz und Nieren überprüft werden. Die Ergebnisse und Erfolgsmeldungen werden jeweils im Browser ausgegeben.

Die Kontrollfunktionen für Objekte und Klassen werden in der Praxis für das Debugging von Programmen eingesetzt. Wenn man das Programm mit Kontrollfunktionen spickt, kann ein möglicher Fehler schnell eingekreist werden.

10 Dateien und Verzeichnisse

PHP verfügt im Umgang mit Dateien und Verzeichnissen über eine große Anzahl von Funktionen, die es erlauben, auf die komplette Datenstruktur eines Servers zuzugreifen. Je nach Betriebssystem des Servers kann es zu einer Einschränkung der Zugriffsrechte aufgrund der Userreglementierung kommen, PHP selbst beschränkt den Einsatz der Dateifunktionen allerdings nicht.

Ein PHP-Skript kann nur auf die Daten des eigenen Servers zugreifen. Die Dateifunktionen erlauben es nicht, auf die Daten eines Clientsystems zuzugreifen und beispielsweise fremde Daten eines anfragenden PCs auszulesen. Da PHP eine serverseitige Sprache ist, sind solche Zugriffe grundsätzlich ausgeschlossen.

PHP ist in der Lage, über eine HTTP-Anfrage Dateien eines anderen Webservers abzufragen und die Daten zu verarbeiten. Dabei handelt es sich selbstverständlich nur um einen reinen Lesezugriff.

Die Mächtigkeit der PHP-Funktionen im Umgang mit dem Dateisystem des Servers kann bei falscher Anwendung zu einem nicht zu unterschätzenden Sicherheitsrisiko werden. Darum sollten Sie darauf achten, in einem Skript niemals direkten Zugriff auf das Dateisystem des Servers zu gestatten. Es ist ohne weiteres möglich, ein Skript zu schreiben, mit dem ein User die Daten der kompletten Webseite zu vernichten kann. Bei falscher Konfiguration des Servers sogar weit mehr.

10.1 Kapitelübersicht

- Grundlegende Dateifunktionen
- Dateien lesen und schreiben
- Weitere Dateifunktionen
- Verzeichnisfunktionen
- Informationen über Dateien und Verzeichnisse
- Dateiupload mit PHP

10.2 Grundlegende Dateifunktionen

Die grundlegenden Dateifunktionen von PHP beschreiben das Öffnen und Schließen von Dateien im jeweiligen Betriebssystem des Servers. Um mit einer Datei arbeiten zu können, muss sie zuerst von PHP geöffnet werden. Eine geöffnete Datei wird (je nach Modus) vom Betriebssystem für andere Anwendungen gesperrt, so dass es unmöglich ist, eine Datei doppelt zu bearbeiten.

Eine Datei, die zum Lesen geöffnet wurde, erhält einen Schreibschutz, damit der Inhalt während des Lesevorgangs nicht verändert werden kann. Eine Datei, die zum Schreiben geöffnet wurde, erhält einen Lese- und Schreibschutz gegenüber anderen Anwendungen. So wird verhindert, dass eine Datei gleichzeitig von zwei Programmen manipuliert werden kann.

PHP öffnet eine Datei mit der Funktion `fopen()`, die zwei Parameter übernimmt. Als Erstes den Dateinamen mit direkter oder relativer Pfadangabe und als Zweites eine Stringkonstante, die den Modus des Zugriffs festlegt. War der Vorgang erfolgreich, dann gibt die Funktion einen Dateihandle zurück, der intern auf die geöffnete Datei zeigt. Dieser Handle muss bei allen Funktionen für die Dateibearbeitung als Parameter angegeben werden.

Der Modus des Dateizugriffs legt fest, wie PHP die Datei öffnen soll und welche Zugriffe über den Dateihandle möglich sind. PHP unterscheidet zwischen einem reinen Lesezugriff, einem Schreibzugriff und der Möglichkeit Daten an eine bestehende Datei anzuhängen. Jeder Modus wird durch einen Buchstaben repräsentiert, die mit einem Pluszeichen kombiniert werden können.

Buchstabe	Dateimodus
`r`	Lesezugriff
`r+`	Lese- und Schreibzugriff
`w`	Schreibzugriff: Der Inhalt der Datei wird gelöscht. Existiert die Datei nicht, dann wird sie erstellt.
`w+`	Lese- und Schreibzugriff: Der Inhalt der Datei wird gelöscht. Existiert die Datei nicht, dann wird sie erstellt.

Tabelle 10.1: Dateizugriffsmodi

Buchstabe	Dateimodus
a	Schreibzugriff: Die Daten werden an das Ende der Datei angehängt. Existiert die Datei nicht, dann wird sie erstellt.
a+	Lese- und Schreibzugriff: Die Daten werden an das Ende der Datei angehängt. Existiert die Datei nicht, dann wird sie erstellt.

Tabelle 10.1: Dateizugriffsmodi (Forts.)

Über das Pluszeichen kann der Zugriffsmodus erweitert werden. Je nach Art des Zugriffs wird von PHP die Datei für die Arbeit vorbereitet: Möchte man in die Datei schreiben, dann wird der Inhalt vor dem Zugriff gelöscht. Im Gegensatz dazu besteht die Möglichkeit, Daten an den bestehenden Inhalt der Datei anzuhängen.

```
$handle = fopen("datei", "r");
```

Nach dem Zugriff auf eine Datei muss diese wieder geschlossen werden. Solange eine Datei geöffnet ist, wird sie vom Betriebssystem gesperrt und andere Applikationen können nicht auf sie zugreifen. Das Schließen einer Datei geschieht über die Funktion `fclose()`, der als Parameter das betreffenden Dateihandle übergeben wird.

```
fclose($handle);
```

Die Funktion `fclose()` befreit das Dateihandle von der bestehenden Funktion, so dass die Variable danach neu gesetzt werden kann.

Das Beispiel zeigt, wie eine Datei geöffnet und danach wieder geschlossen wird. Zwei Kontrollbedingungen überprüfen den Erfolg der einzelnen Funktionen. Für dieses und für alle folgenden Programme gehe ich davon aus, dass im Verzeichnis des Skripts eine Textdatei mit dem Namen `data.txt` existiert.

```
<?
$handle = fopen("data.txt", "r");
if($handle != NULL) {echo "Die Datei wird geöffnet...<br>";}
$bool = fclose($handle);
if($bool) {echo "...und wieder geschlossen!";}
?>
```

Listing 10.1: Eine Datei wird geöffnet und wieder geschlossen

Ausgabe:

```
Die Datei wird geöffnet...
...und wieder geschlossen!
```

Das Programm ist sicher nicht sehr sinnvoll, aber auf dieser Grundlage können wir alle weiteren Dateifunktionen nutzen.

Ein großer Vorteil von PHP ist der intelligente Umgang mit Dateien, die nicht auf dem eigenen Server liegen. Die Funktion `fopen()` erkennt automatisch, dass es sich bei der zu öffnenden Datei um eine URL handelt, und stellt selbstständig eine HTTP-Verbindung zum angegebenen Server her. Dabei spielt es keine Rolle, ob es sich um ein HTML-Dokument oder eine einfache Textdatei handelt.

Wenn die angegebene Datei gefunden wurde, dann gibt `fopen()` wie gewohnt ein Dateihandle zurück, das es erlaubt, auf die Datei zuzugreifen. Die Rechte beschränken sich allerdings nur auf einen reinen Lesezugriff.

```
<?
$handle = fopen("http://www.kulturbrand.de/index.html", "r");
if($handle != NULL) {echo "Die Datei wurde geöffnet...<br>";}
$bool = fclose($handle);
if($bool) {echo "...und wieder geschlossen!";}
?>
```

Listing 10.2: Zugriff auf die Dateien eines anderen Servers

Ausgabe:

```
Die Datei wurde geöffnet...
...und wieder geschlossen!
```

Das Skript öffnet die Datei `index.html` auf dem Server *`kulturbrand.de`* und schließt sie danach wieder. Voraussetzungen für dieses Skript sind natürlich der Kontakt zum Internet sowie die Existenz der URL.

Der Dateizeiger

PHP setzt ähnlich wie bei Arrays einen Zeiger ein, um die aktuelle Lese-/Schreib-Position in einer Datei zu kennzeichnen. Die Technik ist vergleichbar mit dem Cursor eines Schreibprogramms, der die Stelle markiert, an der der Text bearbeitet werden kann.

Je nach Zugriffsmodus sitzt der Dateizeiger an einer anderen Startposition: Bei Lese- und Schreibzugriffen mit den Parametern **s** oder **w** wird der Zeiger an die Position Null (an den Anfang) der Datei gesetzt. Der Zugriff mit **a** legt die Startposition des Zeigers hinter das letzte Zeichen der Datei.

Immer wenn eine Datei gelesen oder bearbeitet wird, ändert sich die Position des Dateizeigers. Je nach Funktion wird der Zeiger um einzelne Zeichen oder gleich um ganze Zeilen verschoben. PHP verfügt über verschiedene Funktionen um die Position des Dateizeigers zu erfragen beziehungsweise zu manipulieren.

Funktion	Beschreibung
`ftell($handle)`	gibt die aktuelle Position des Dateizeigers als Ganzzahl zurück
`rewind($handle)`	setzt den Dateizeiger an den Anfang der Datei
`fseek($handle, $num)`	setzt den Dateizeiger um eine festgelegt Anzahl von Zeichen vor oder zurück
`feof($handle)`	prüft, ob der Dateizeiger am Ende der Datei ist

Tabelle 10.2: Funktionen zur Manipulation des Dateizeigers

Jede der Funktionen muss das Dateihandle der Zieldatei als Parameter übergeben bekommen. Die Funktion `fseek()` braucht zusätzlich eine positive oder negative Ganzzahl, die die Vor- oder Rückschritte des Dateizeigers festlegt.

```
<?
$h = fopen("data.txt", "r");
echo ftell($h) ."<br>";
fseek($h,100);
echo ftell($h) ."<br>";
rewind($h);
echo ftell($h);
fclose($h);
?>
```

Listing 10.3: Manipulation des Dateizeigers

Ausgabe:

```
0
100
0
```

Die Funktion `rewind()` bietet sich an, wenn die Gefahr besteht sich in einer Datei zu »verlaufen«. Sie setzt den Dateizeiger auf die Ausgangsposition der Datei, so dass man sich neu orientieren kann.

10.3 Dateien lesen und schreiben

PHP unterstützt eine Reihe von einfachen Funktionen, die sich auf das Lesen und Schreiben von Dateien beziehen. Alle Funktionen brauchen das Dateihandle der Zieldatei, um auf sie zugreifen zu können.

Funktion	Beschreibung
`fgetc($h)`	Liest das nächste Zeichen nach dem Dateizeiger und gibt es zurück.
`fgets($h, $num)`	Liest eine bestimmte Anzahl von Zeichen aus der Zeile, in der der Dateizeiger steht, und gibt sie zurück.
`fpassthru($h)`	Liest die gesamte Datei und gibt den Inhalt mit der Anzahl der Zeichen zurück.
`fputs($h, $str)`	Schreibt einen String an die Stelle nach dem Dateizeiger.
`fwrite($h, $str)`	wie fputs()
`fread($h, $num)`	Liest eine bestimmte Anzahl von Zeichen nach dem Dateizeiger ein.

Tabelle 10.3: Funktionen zur Dateibearbeitung

Je nach Situation und Bedarf kann eine Datei Zeichen für Zeichen oder in einem Stück ausgelesen werden. Jede Möglichkeit hat ihre Vor- und Nachteile.

Möchte man eine Datei in einem Stück auslesen, dann ist die Funktion `fpassthru()` sicherlich die beste Wahl. Sie gibt den gesamten Inhalt der Datei in einem Stück als String zurück und braucht keine weiteren Angaben als das Dateihandle der Zieldatei.

```
<?
$handle = fopen("data.txt", "r");
$str = fpassthru($handle);
echo $str;
?>
```

Listing 10.4: Die Funktion fpassthru()

Ausgabe:

```
Und eines Tages hatten sie es satt, für andere Geld reinzuholen und
dunkle Limousinen zu fahren. Sie wollten keine Sonnenbrillen mehr
tragen und elegante Lederkoffer an geheimen Orten übergeben, um die
Welt vor neuen Kriegen zu retten. Vorbei waren die Zeiten von durch-
feierten Nächten und luxuriösen Jachten, auf denen immer neue Schön-
heiten auf sie warteten um sich zu vergnügen. Sie wollten endlich
selbstständig sein und auf eigenen Beinen stehen. Es war genau die
richtige Zeit für eine Änderung. Und als die Chance kam, ergriffen
sie diese. Eine neue Zeit brach an... 582
```

Die Ausgabe des Skript ist keine große Überraschung, da wir den Inhalt der Datei schon kennen. Interessanter dagegen ist die Zahl ganz am Ende des Textes. Sie gibt an, wie viele Zeichen der übergebene String hat, und erlaubt es so, die Größe der Datei zu schätzen. Da jedes Zeichen ein Byte Speicherplatz in Anspruch nimmt (ASCII-Codierung vorausgesetzt), hat die Datei eine Größe von 582 Byte.

Eine weitere nützliche Eigenschaft der Funktion `fpassthru()` ist das selbstständige Schließen der Datei. Die Funktion übernimmt die Aufgabe der Funktion `close()` und räumt sozusagen hinter sich auf.

Möchte man es etwas genauer haben, kann man eine Datei auch zeichen- oder zeilenweise auslesen. Der Vorgang ist bei beiden Techniken ähnlich, da man in der Regel nicht weiß, wie groß eine Datei ist. Um nicht über das Ende der Datei hinauszulesen, bemüht man eine Schleifenkonstruktion, die nach jeder Zeile überprüft, wie weit das Skript ist.

```
<?
$handle = fopen("data.txt", "r");
$i=1;
while(!feof($handle))
{
        $str = fgets($handle, 200);
        echo "$i: $str<br>";
        $i++;
}
fclose($handle);
?>
```

Listing 10.5: Eine Datei wird zeilenweise ausgelesen

Ausgabe:

```
1: Und eines Tages hatten sie es satt, für andere Geld reinzuholen
2: und dunkle Limousinen zu fahren. Sie wollten keine Sonnenbrillen
3: mehr tragen und elegante Lederkoffer an geheimen Orten übergeben,
4: um die Welt vor neuen Kriegen zu retten. Vorbei waren die Zeiten
5: von durchfeierten Nächten und luxuriösen Jachten, auf denen immer
6: neue Schönheiten auf sie warteten um sich zu vergnügen. Sie woll-
ten
7: endlich selbstständig sein und auf eigenen Beinen stehen. Es war
8: genau die richtige Zeit für eine Änderung. Und als die Chance kam,
9: ergriffen sie diese. Eine neue Zeit brach an...
```

Der Kern des Skripts ist eine `while()`-Schleife, mit der nach jeder Zeile überprüft wird, ob das Dateiende schon erreicht wurde. Solange die Funktion `feof()` FALSE zurückgibt, wird weitergelesen. Diese Vorgehen erlaubt es, zwischen den einzelnen Zeilen der Datei genau zu differenzieren. So ist es zum Beispiel möglich, die Formatierung der Datei zu übernehmen oder eine durchgehende Zeilennummerierung einzufügen.

In Dateien schreiben

Die Funktionen `fputs()` und `fwrite()` sind im Prinzip identisch, da beide die Aufgabe haben, Strings in eine Datei zu schreiben. Der String wird immer hinter der aktuellen Position des Dateizeigers eingefügt, d.h. wenn mehrere Strings hintereinander eingetragen werden, dann werden sie nicht überschrieben.

```
<?
$handle = fopen("data.txt", "w");
$bool=fputs($handle, "Dies ist eine Zeile in einer Textdatei!");
if ($bool) {echo "String wurde eingetragen...";}
fclose($handle);
?>
```

Listing 10.6: Ein String wird in eine Datei geschrieben

Ausgabe:

```
String wurde eingetragen...
```

Ein Blick in die Textdatei offenbart das Ergebnis dieses Programms. Sie enthält jetzt die Zeile:

```
Dies ist eine Zeile in einer Textdatei!
```

Wenn Sie sich jetzt fragen, was mit dem alten Text geschehen ist, der in der Datei gespeichert war, dann bedenken Sie bitte, dass die Datei mit »w« geöffnet worden ist. In diesem Modus wird der komplette Inhalt der Datei gelöscht, bevor sie zur Bearbeitung freigegeben wird.

Im Gegensatz dazu erlaubt der Parameter »a« Strings an eine Datei anzuhängen. Der Inhalt der Datei wird nicht gelöscht und der Dateizeiger steht hinter dem letzten Zeichen.

```
<?
$handle = fopen("data.txt", "a");
$bool=fwrite($handle, " Dieser Text wird angehängt!");
if ($bool) {echo "String wurde eingetragen...";}
fclose($handle);
?>
```

Listing 10.7: Text wird angehängt

Ausgabe:

```
String wurde eingetragen...
```

Zur Abwechslung wurde diesmal die Funktion `fwrite()` verwendet, um die Datei zu bearbeiten. Ein neuer Blick in data.txt zeigt das Ergebnis des Skripts:

```
Dies ist eine Zeile in einer Textdatei! Dieser Text wird angehängt!
```

Der String wurde an den bestehenden Inhalt der Datei angehängt. Jede weitere Ausführung des Programms würde dazu führen, dass immer wieder ein neuer String an die Textdatei angehängt wird.

Um eine Textdatei formatieren zu können, brauchen Sie verschiedene Sonderzeichen, die es erlauben, Steuerzeichen in Dateien zu setzen. Die wichtigsten sind das Symbol für den Zeilenumbruch sowie für die Tabulatortaste.

Steuerzeichen	Symbol
`\ n`	Zeilenumbruch
`\ t`	Tabulator

Tabelle 10.4: Steuerzeichen

Steuerzeichen können einfach in einen String geschrieben werden, da PHP sie automatisch interpretiert.

```
<?
$handle = fopen("data.txt", "w");
fwrite($handle, " Dies ist eine Zeile!\n");
fwrite($handle, " Dies ist eine Zeile \t mit Tabulator!\n");
fclose($handle);
?>
```

Listing 10.8: Sonderzeichen

Die Textdatei hat nach Ausführung des Programms folgenden Inhalt:

```
Dies ist eine Zeile!
Dies ist eine Zeile  mit Tabulator!
```

Einige Editoren unter Windows sind unter Umständen nicht in der Lage, die gesetzten Sonderzeichen korrekt zu interpretieren, da hier auf eine UNIX-Codierung zurückgegriffen wird. Wenn Sie das Programm Wordpad verwenden, sollten Sie aber keine Probleme haben.

10.4 Weitere Dateifunktionen

Neben den einfachen Funktionen zum Lesen und Schreiben von Dateien bietet PHP noch eine Reihe von erweiterten Techniken, die den Umgang mit Dateien erleichtern. Dazu gehören unter anderem Funktionen zum Kopieren, Umbenennen und Löschen von Dateien sowie weitere Methoden um eine Datei auszulesen.

Im Gegensatz zu den vorher besprochenen Möglichkeiten benötigen diese Funktionen keinen Dateihandle, um mit einer Datei zu arbeiten. Es ist also nicht nötig, die Zieldatei zu öffnen oder zu schließen.

Funktion	Beschreibung
`file("Dateiname")`	Liest den kompletten Inhalt der Datei ein und gibt ihn als Array zurück
`readfile("Dateiname")`	Liest eine Datei und gibt sie wieder aus
`copy("Quelle", "Ziel")`	Kopiert eine Datei
`rename("Name", "NeuName")`	Benennt eine Datei um
`unlink("Dateiname")`	Löscht eine Datei

Tabelle 10.5: weitere Dateifunktionen

Die neuen Funktionen sind eigentlich selbsterklärend und bedürfen keiner großen Beispiele. Das folgende Skript zeigt, wie damit gearbeitet wird.

```
<?
copy("data.txt", "datei.bak");
readfile("datei.bak");
unlink("data.txt");
rename("datei.bak", "data.txt");
?>
```

Listing 10.9: Eine Datei wird bearbeitet

Ausgabe:

```
Und eines Tages hatten sie es satt, für andere Geld reinzuholen und
dunkle Limousinen zu fahren. Sie wollten keine Sonnenbrillen mehr
tragen und elegante Lederkoffer an geheimen Orten übergeben, um die
Welt vor neuen Kriegen zu retten. Vorbei waren die Zeiten von durch-
feierten Nächten und luxuriösen Jachten, auf denen immer neue Schön-
heiten auf sie warteten um sich zu vergnügen. Sie wollten endlich
selbstständig sein und auf eigenen Beinen stehen. Es war genau die
richtige Zeit für eine Änderung. Und als die Chance kam, ergriffen
sie diese. Eine neue Zeit brach an...
```

Das Skript besteht aus vier einfachen Funktionsaufrufen, die sich auf die Datei data.txt beziehen. Im ersten Schritt wird durch die Funktion `copy()` eine Kopie der vorhandenen Daten erstellt, die unter dem Dateinamen datei.bak abgelegt werden. Zur Kontrolle wird mit der Funktion `readfile()` der Inhalt der Datei ausgegeben und im Browser angezeigt. Mit der Funktion `unlink()`, die übrigens aus dem Repertoire von Perl übernommen wurde, wird die Originaldatei gelöscht, um sie im letzten Schritt über die Funktion `rename()` wieder herzustellen.

Das Skript hat sich also einmal im Kreis gedreht und ist genau da wieder angekommen, wo es begonnen hat. Nicht sehr aufregend, aber trotz allem eine schöne Demonstration der benutzten Funktionen.

Die Funktion `file()` nimmt eine besondere Stellung unter den Dateifunktionen ein, da sie mehrere Aufgaben miteinander verknüpft. Mit `file()` ist es möglich, eine Datei zeilenweise auszulesen und in einem Array zu speichern, ohne sie über `fopen()` öffnen zu müssen. Die Funktion wurde eingeführt, weil in der programmiertechnischen Praxis das zeilenweise Auslesen zu den häufigsten Vorgängen im Umgang mit Dateien gehört.

Das Ergebnisarray ist nummerisch und enthält pro Element eine Zeile der Zieldatei.

```
<?
$liste = file("data.txt");

foreach($liste as $line)
{
        echo $line . "<br>";
}
?>
```

Listing 10.10: Die Funktion file()

Ausgabe:

```
Und eines Tages hatten sie es satt, für andere Geld reinzuholen
und dunkle Limousinen zu fahren. Sie wollten keine Sonnenbrillen
mehr tragen und elegante Lederkoffer an geheimen Orten übergeben,
um die Welt vor neuen Kriegen zu retten. Vorbei waren die Zeiten
von durchfeierten Nächten und luxuriösen Jachten, auf denen immer
neue Schönheiten auf sie warteten um sich zu vergnügen. Sie wollten
endlich selbstständig sein und auf eigenen Beinen stehen. Es war
genau die richtige Zeit für eine Änderung. Und als die Chance kam,
ergriffen sie diese. Eine neue Zeit brach an...
```

Das Ergebnisarray der Funktion wird über eine `foreach()`-Schleife elementweise ausgewertet und formatiert ausgegeben. Der Unterschied zur Funktion `readfile()` besteht darin, dass der Programmierer die Kontrolle über die Daten der Datei behält, bevor sie ausgegeben werden.

10.5 Verzeichnisfunktionen

Die Funktionen und Methoden, die in diesem Kapitel vorgestellt werden, ermöglichen die Arbeit mit Verzeichnissen auf dem Rechner. Je nach verwendetem Betriebssystem auf dem Server müssen die Angaben zu den Zielverzeichnissen angepasst werden. Wesentlich hierbei ist der Unterschied zwischen dem UNIX-Dateisystem und der Verzeichnisverwaltung von Microsoft Windows.

Die Verzeichnisstruktur von Windows dürfte den meisten bekannt sein, da Windows wohl das häufigste Betriebssystem auf Home-PCs ist. Die Grundlage der Verzeichnisstruktur basiert auf den verschiedenen Laufwerken des lokalen PCs, die durch Großbuchstaben gekennzeichnet sind. Die Festplatte mit dem Betriebssystem hat in der Regel die Buchstabenkennung C:

Jedes Verzeichnis wird durch einen Backslash getrennt an den Laufwerksbuchstaben angehängt. Eine typische Verzeichnisangabe könnte unter Windows also so aussehen:

```
C:\Programme\Apache Group\Apache\htdocs\gotoPHP4
```

Das angegebene Verzeichnis heißt gotoPHP4 und liegt in einer Reihe von Unterverzeichnissen auf dem Laufwerk C. Wie zukünftige Programme noch zeigen werden, liegen hier einige Beispielprogramme zu diesem Buch gespeichert.

Im Gegensatz zu Windows verfolgt UNIX in seiner Verzeichnisstruktur eine andere Systematik. Das Ursprungsverzeichnis des Systems bezieht sich nicht auf ein Laufwerk, sondern auf ein so genanntes Root-Verzeichnis, das alle weiteren Verzeichnisse und Laufwerke beinhaltet. Die Struktur ist grob vergleichbar mit dem Desktop der Windowsoberfläche, das ebenfalls alle Daten beinhaltet.

Das Root-Verzeichnis unter UNIX wird mit einem einfachen Slash gekennzeichnet. Jedes weitere Verzeichnis muss vom Root-Verzeichnis ausgehen:

```
/usr/local/apache/htdocs/gotoPHP4
```

Dieses UNIX-Verzeichnis wäre das genaue Gegenstück zum oben angegebenen Windowsverzeichnis mit den Programmbeispielen.

Da sich 99 Prozent aller PHP-Interpreter im Internet auf UNIX oder Linux-Rechnern befinden, werde ich alle folgenden Beispiele in der UNIX-Syntax angeben. Die Programme funktionieren genauso unter Windows mit dem Unterschied, dass die Verzeichnisangaben angepasst werden müssen.

Um bei Bedarf die Kompatibilität des Programms herstellen zu können, bietet es sich an die betroffenen Verzeichnisse am Anfang des Skripts als Konstante oder String-Variable zu definieren. Dadurch ist es möglich, alle Änderungen zentral auszuführen.

10.5.1 Verzeichnisfunktionen unter PHP

Die Verzeichnisfunktionen unter PHP gehören zu dem Bereich der Sprache, der mit objektorientierten Methoden gelöst wurde. Anstatt eine große Anzahl von verschiedenen Funktionen zum Öffnen und Lesen von

Verzeichnissen zu konstruieren, kommt PHP mit drei verschiedenen Funktionen aus. Zwei der Funktionen dienen dem Erstellen und Löschen von Verzeichnissen.

Funktion	Beschreibung
`dir("Verzeichnis")`	erstellt ein Verzeichnisobjekt und gibt es zurück
`mkdir("Verzeichnis", $modus)`	erstellt ein Verzeichnis
`rmdir("Verzeichnis")`	löscht ein Verzeichnis

Tabelle 10.6: Verzeichnisfunktionen unter PHP

Die Funktion `dir()` öffnet ein Verzeichnis und erstellt ein Objekt, das zu ihrer weiteren Verarbeitung dient. Dieses Verzeichnisobjekt wird automatisch von der Funktion zurückgegeben, wenn das Öffnen des Zielverzeichnisses erfolgreich war.

```
$obj = dir("/usr/local/apache/htdocs/gotoPHP4");
```

Das Objekt sollte in einer Variable gespeichert werden, damit der Zugriff auf die verfügbaren Objektmethoden und Variablen möglich ist. PHP unterstützt drei verschiedene Zugriffsmethoden und zwei weitere Eigenschaftsvariablen.

Methode / Variable	Beschreibung
`$obj - > read()`	gibt den nächsten Eintrag im Verzeichnis zurück. Gibt FALSE zurück, wenn kein Eintrag mehr vorhanden ist
`$obj -> rewind()`	setzt den Verzeichniszeiger auf den Anfang des Verzeichnisses zurück
`$obj -> close()`	schließt das Verzeichnis und zerstört das Verzeichnisobjekt
`$obj -> handle`	gibt ein Handle auf das Verzeichnis zurück
`$obj -> path`	gibt den Pfad zurück

Tabelle 10.7: Objektmethoden und Variablen von Verzeichnisobjekten

Genau wie bei Dateien arbeitet PHP mit einem so genannten Verzeichniszeiger, der auf den aktuellen Eintrag eines Verzeichnisses zeigt. Das Ziel kann ein Verweis sein, eine Datei oder ein weiteres Verzeichnis. Jeder Auf-

ruf der Funktion `read()` setzt den Verzeichniszeiger einen Eintrag weiter. Die Funktion `rewind()` setzt den Zeiger wieder an den Anfang des Verzeichnisses.

```
<?
$obj = dir("/usr/local/apache/htdocs/gotoPHP4");
echo "Inhalt vom Verzeichnis: <b>";
echo $obj ->path . "<br></b>";

while($var=$obj -> read())
{
        echo $var  . "<br>";
}
$obj -> close();
?>
```

Listing 10.11: Die Verzeichnisfunktion dir()

Ausgabe:

```
Inhalt vom Verzeichnis:/usr/local/apache/htdocs/gotoPHP4
.
..
gtest.php
data.txt
listing123.php
listing122.php
listing124.php
listing125.php
listing126.php
listing121.php
listing127.php
scripte
```

Das Skript liest ein Verzeichnis aus, in dem ein Teil der Beispielskripte für dieses Buch gespeichert sind. Die Ausgabe der Verzeichnisbestandteile beginnen mit den Verweisen auf das eigene Verzeichnis sowie auf das darüber liegende Verzeichnis, die jeweils mit Punkten symbolisiert werden. Danach folgt eine Liste der vorhandenen Dateien sowie ein weiteres Unterverzeichnis.

Mit den so gewonnenen Informationen ist es möglich, im Verzeichnissystem des Rechners zu navigieren und nach bestimmten Daten zu fahnden.

Um eigene Verzeichnisse zu erstellen beziehungsweise vorhandene Datenverzeichnisse zu löschen, brauchen Sie zwei weitere Funktionen, die bereits

in der oberen Tabelle vorgestellt wurden. Die Funktion `mkdir()` erlaubt Verzeichnisse, die unter UNIX sogar mit Informationen über die Zugriffsrechte versehen werden können. Nähere Informationen zu den Zugriffsrechten unter UNIX bekommen Sie im übernächsten Kapitel. Unter Windows haben die Angaben keinerlei Bedeutung und werden ignoriert.

```
mkdir("test", 0777);
```

Dieser Befehl erschafft ein neues Verzeichnis im selben Verzeichnis, in dem das ausführende Skript liegt. Das Verzeichnis heißt test und hat unter UNIX keinerlei Zugriffsbeschränkungen.

Das Gegenstück zu `mkdir()` ist die Funktion `rmdir()`, die vorhandene Verzeichnisse löscht. Als einzigen Parameter benötigt die Funktion den Namen des Zielverzeichnisses aus String.

```
rmdir("test");
```

Dieser Aufruf löscht das oben erzeugte Verzeichnis und gibt bei Erfolg TRUE zurück. Bitte beachten Sie, dass die Funktion nur leere Verzeichnisse löschen kann. Enthält das Zielverzeichnis Dateien, müssen die zuerst manuell gelöscht werden.

Genau wie bei Dateien ist es nötig, dass man für den Einsatz der Verzeichnisfunktionen ausreichende Rechte auf dem Server hat. PHP gibt eine Fehlermeldung zurück, wenn das Betriebssystem den Zugriff auf Verzeichnisse verweigert.

10.5.2 Informationen über Dateien und Verzeichnisse

Um mit Dateien und Verzeichnissen optimal arbeiten zu können, sind einige Informationen über die Daten eines Dateisystems notwendig. PHP unterstützt eine ganze Reihe von Funktionen, mit denen Informationen über Dateien und Verzeichnisse abgefragt bzw. überprüft werden können.

Funktion	Beschreibung
`diskfreespace($pfad)`	gibt den freien Speicherplatz im Zielverzeichnis zurück
`filesize($dateiname)`	gibt die Größe einer Datei zurück
`filetype($dateiname)`	gibt den Dateityp zurück

Tabelle 10.8: Informationsfunktionen für Dateien und Verzeichnisse

Funktion	Beschreibung
`filectime($dateiname)`	gibt den Zeitpunkt der Dateierstellung zurück
`filemtime($dateiname)`	gibt den Zeitpunkt der letzten Dateiänderung zurück
`fileatime($dateiname)`	gibt den Zeitpunkt des letzten Dateizugriffs zurück
`file_exits($dateiname)`	überprüft, ob die angegebene Datei existiert
`is_file($dateiname)`	überprüft, ob das angegebene Ziel eine Datei ist
`is_dir($verzeichnis)`	überprüft, ob das angegebene Ziel ein Verzeichnis ist
`is_link($link)`	überprüft, ob das angegebene Ziel ein Link ist
`is_readable($ziel)`	überprüft, ob das angegebene Ziel lesbar ist
`is_writable($ziel)`	überprüft, ob das angegebene Ziel beschreibbar ist
`touch($dateiname, $tstamp)`	ändert das Zugriffsdatum einer Datei

Tabelle 10.8: Informationsfunktionen für Dateien und Verzeichnisse (Forts.)

Um die neuen Funktionen zu demonstrieren, wird das Listing 10.11 aus dem letzten Kapitel ein wenig erweitert. Das Programm soll jetzt nicht nur den Inhalt des Verzeichnisses liefern, sondern auch eine Reihe von nützlichen Informationen zu den einzelnen Elementen.

Damit auch Windows mal zu Wort kommt, habe ich dieses Programm für einen Windowsrechner geschrieben:

```
<?
$obj = dir("C:\Programme\Apache Group\Apache\htdocs\gotoPHP4");
echo "Inhalt vom Verzeichnis: <b>";
echo $obj ->path . "</b> (" . diskfreespace($obj ->path) . " Byte
frei)<br><br>";
while($var=$obj -> read())
{
        echo "$var   ". filesize($var) ." Byte, ";
        echo filetype($var) . " (erstellt am " . date("d.M Y H:i:s",
filectime($var)) . "), ";
        if (is_readable($var)) {echo "Leserechte, ";}
        if (is_writable($var)) {echo "Schreibrechte";}
        echo "<br>";
}
$obj -> close();
?>
```

Listing 10.12: Informationen über Dateien und Verzeichnisse

Ausgabe:

```
Inhalt vom Verzeichnis: C:\Programme\Apache Group\Apa-
che\htdocs\gotoPHP4 (11290476544 Byte frei)

. 0 Byte, dir (erstellt am 28.Sep 2001 14:05:13), Leserechte,
Schreibrechte
.. 0 Byte, dir (erstellt am 04.Jan 2001 02:51:37), Leserechte,
Schreibrechte
gtest.php 808 Byte, file (erstellt am 03.Nov 2001 14:08:52), Lese-
rechte, Schreibrechte
listing128.php 493 Byte, file (erstellt am 08.Nov 2001 18:01:40),
Leserechte, Schreibrechte
listing115.php 213 Byte, file (erstellt am 03.Nov 2001 20:38:56),
Leserechte, Schreibrechte
listing116.php 785 Byte, file (erstellt am 04.Nov 2001 14:50:19),
Leserechte,
listing127.php 217 Byte, file (erstellt am 06.Nov 2001 22:34:46),
Leserechte, Schreibrechte
listing15.php 76 Byte, file (erstellt am 30.Sep 2001 17:10:10), Lese-
rechte, Schreibrechte
listing16.php 113 Byte, file (erstellt am 30.Sep 2001 17:20:13),
Leserechte,
listing20.php 169 Byte, file (erstellt am 30.Sep 2001 22:20:43),
Leserechte,
```

Das Programm macht im Prinzip nichts anderes als die vorhergehende Version aus dem letzten Kapitel. Es wurde nur eine Reihe von neuen Funktionen hinzugefügt, die jedes Element des Verzeichnisses überprüfen. Wir erfahren unter anderem, wie viel Speicherplatz jede Datei braucht, wann sie erstellt wurden und welche Rechte bezüglich der Datei vorhanden sind.

Die Funktionen `filectime()`, `fileatime()` und `filemtime()` geben den gewünschten Zeitpunkt als UNIX-Timestamp zurück. Um diese Zahlenkombination in ein lesbares Format umzuwandeln, braucht man die Funktion `date()`, die wir bereits in einem der letzten Kapitel kennen gelernt haben.

10.5.3 Weitere Funktionen unter UNIX

Jede Datei und jedes Verzeichnis unter UNIX/Linux unterliegt einem strengen Reglement, das bestimmt, welcher User auf welche Daten zugreifen darf. Diese und weitere Informationen werden für alle Dateien und Verzeichnisse in einer mehrstelligen Ganzzahl gespeichert, die man

über PHP erfragen kann. Der vordere Teil dieser Zahl beschreibt den Typ des Zielobjekts und der hintere Teil die Zugriffsrechte.

Über die Funktion `fileperms()` kann PHP den Wert dieser Zahl abfragen.

```
$num = fileperms("data.txt");
```

Um den Wert korrekt interpretieren zu können ist es allerdings notwendig, die Zahl in eine Oktalzahl umzuwandeln. Das geschieht mit der Funktion `decoct()`, die wir bereits im Kapitel über mathematische Funktionen kennen gelernt haben.

```
<?
$num = fileperms("data.txt");
echo decoct($num);
?>
```

Listing 10.13: Zugriffsrechte unter UNIX

Ausgabe:

```
100644
```

Die ersten drei Stellen der Oktalzahl (100) geben an, dass es sich beim Zielobjekt um eine einfache Datei handelt. Die letzten drei Stellen (644) beschreiben die Zugriffsrechte dieser Datei unter UNIX.

UNIX unterscheidet drei verschiedene Möglichkeiten des Zugriffs auf eine Datei. Es gibt den einfachen Lesezugriff (read), den Schreibzugriff (write) und als höchste Priorität das Recht zum Ausführen (execute) der Datei. Jeder dieser Zustände wird durch eine Zahl beschrieben, die bei einer Kombination von Rechten einfach zusammengerechnet werden.

Zahl	Rechte
1	Ausführungsrechte von Programmen und Skripten
2	Schreibrechte
4	Leserechte

Tabelle 10.9: Zugriffsrechte unter UNIX

Gleichzeitig unterscheidet UNIX drei verschiedene Gruppen von Benutzern. Die erste Gruppe, die in der Regel immer die höchsten Zugriffsrechte hat, sind die Besitzer der Datei. Ihre Rechte werden in der ersten Ziffer des dreistelligen Codes angegeben.

Die zweite Gruppe besteht aus allen Mitgliedern der Benutzergruppe des Besitzers. Ihre Rechte werden durch die zweite Ziffer des Codes angegeben.

Die dritte Gruppe unter UNIX ist der sprichwörtliche Rest im Computeruniversum. Jeder User, der Zugriff auf das System hat und nicht zu den ersten beiden Gruppen gehört, bekommt die Rechte auf die Datei zugewiesen, die in der letzten Stelle des Zugriffscodes definiert wurden.

Die Zugriffsrechte der Datei data.txt aus dem obigen Beispiel (644) legen also fest, dass der Besitzer der Datei Lese- und Schreibrechte besitzt (4 + 2) und dass die Benutzergruppe sowie der ganze Rest nur Leserechte an der Datei haben (4).

Weitere typische Festlegungen für Zugriffsrechte unter UNIX sind:

- 755: Lese- und Ausführungsrechte für alle; nur der Besitzer hat Schreibzugriff
- 700 oder 600: Ausschluss aller User des Systems
- 777: kompletter Zugriff für alle User

Die Funktion `chmod()` erlaubt es, die Zugriffsrechte einer Datei über PHP zu verändern. Dazu sind zwei Parameter nötig: einmal der Dateiname der Zieldatei und als Zweites die neuen Zugriffsrechte.

```
<?
$num = fileperms("data.txt");
echo "Die Zugriffsrechte vor chmod(): " . decoct($num);
chmod("data.txt", 755);
$num = fileperms("data.txt");
echo "<br>Die Zugriffsrechte nach chmod(): " . decoct($num);
?>
```

Listing 10.14: Die Funktion chmod()

Ausgabe:

```
Die Zugriffsrechte vor chmod(): 100644
Die Zugriffsrechte nach chmod(): 100755
```

Damit der Code für die neuen Zugriffsrechte erkannt wird, muss eine Null vorangestellt werden.

Eine weitere Spezialität unter UNIX ist die Möglichkeit, den Besitzer einer Datei explizit festlegen zu können. Über die Funktion `fileowner()`

kann der Name des Besitzers ausgelesen werden. Um den Namen zu ändern, braucht man die Funktion `chown()`, die für eine bestimmte Datei einen neuen Besitzer festlegt.

```
<?
echo "Besitzer vor chown(): " . fileowner("data.txt");
chown("data.txt", "root");
echo "<br>Besitzer nach chown(): " . fileowner("data.txt");
?>
```

Listing 10.15: Die Funktionen fileowner() und chown()

Ausgabe:

```
Besitzer vor chown(): aburg
Besitzer nach chown(): root
```

Um die Funktion `chown()` benutzen zu können, brauchen Sie Administratorrechte auf dem System. Alle hier vorgestellten Funktionen haben unter Windows keinen Effekt und sind sinnlos.

10.6 Dateiupload mit PHP

PHP unterstützt Dateiuploads mit allen Browsern, die dem Standard RFC 1867 entsprechen. Dazu gehören der Netscape Navigator ab der Version 3 und der Microsoft Explorer standardmäßig ab der Version 4. Dieser so genannte HTTP-Transfer vom lokalen Rechner auf den Server kann sowohl mit Textdateien als auch mit Binärcode durchgeführt werden.

PHP gibt dem Programmierer die Möglichkeit genau festzulegen, wer welche Dateien hochladen darf und von welcher Größe die Daten sein dürfen. Dazu steht ihm der komplette Funktionsumfang der PHP-Dateifunktionen zur Verfügung. Voraussetzung für den Upload von Dateien ist die korrekte Konfiguration der php.ini. In dieser Datei kann festgelegt werden, ob und in welchem Umfang der Upload von Dateien gestattet ist.

Wesentlich für die Konfiguration von Dateiuploads sind die folgenden Einträge:

```
file_uploads=On
upload_tmp_dir=c:\php\uploadtemp
upload_max_filesize=2M
```

Die erste Zeile legt fest, ob es generell erlaubt ist, Dateien per POST auf dem Server zu laden. Der zweite Punkt legt das Verzeichnis fest, in dem

die Dateien während der Laufzeit des Programms gespeichert werden sollen. Dieses Verzeichnis wird automatisch geleert, wenn das Skript abgeschlossen wurde. Die letzte Zeile legt die maximale Größe einer Datei fest, die hochgeladen werden darf.

Fragen Sie bei Ihrem Systemadministrator nach, wie die aktuelle Konfiguration Ihres Servers aussieht und ob der Upload von Dateien erlaubt ist. Wenn Sie selbst einen Server konfigurieren wollen, dann lesen Sie das Kapitel über »Das Arbeitsumfeld« für weitere Informationen.

10.6.1 Das Formular

Voraussetzung für den Upload von Dateien ist ein HTML-Formular, das den INPUT-Typ *file* akzeptiert. Das wird durch ein neues Attribut im FORM-Tag gewährleistet, indem der ENCTYPE auf den Wert `multipart/form-data` gesetzt wird.

```
<FORM ENCTYPE="multipart/form-data" ACTION="listing132.php"
METHOD="POST">
```

Wesentlich ist, dass die POST-Methode zur Übertragung der Daten angegeben wird, da die GET-Methode den Upload von Daten nicht leisten kann.

Im nächsten Schritt muss ein INPUT-Feld generiert werden, über das die Zieldatei ausgewählt werden kann. Das geschieht über den TYPE-Parameter `file`, der ein spezielles Eingabefeld schafft.

```
<INPUT NAME="datei" TYPE="file">
```

Der Browser erkennt automatisch, dass der User eine Datei auswählen muss, und schafft ein komfortables Auswahlmenü, über das die Datei bequem erreicht werden kann. Es öffnet sich einfach ein weiteres Fenster, mit dem man sich durch das lokale Verzeichnissystem klickt.

Hier ist der komplette HTML-Code der Webseite zum Upload einer Datei:

```
<FORM ENCTYPE="multipart/form-data" ACTION="listing132.php"
METHOD=POST>
Wählen Sie eine Datei:<br>
<INPUT NAME="datei" TYPE="file"><BR>
<INPUT TYPE="submit" VALUE="Upload">
</FORM>
```

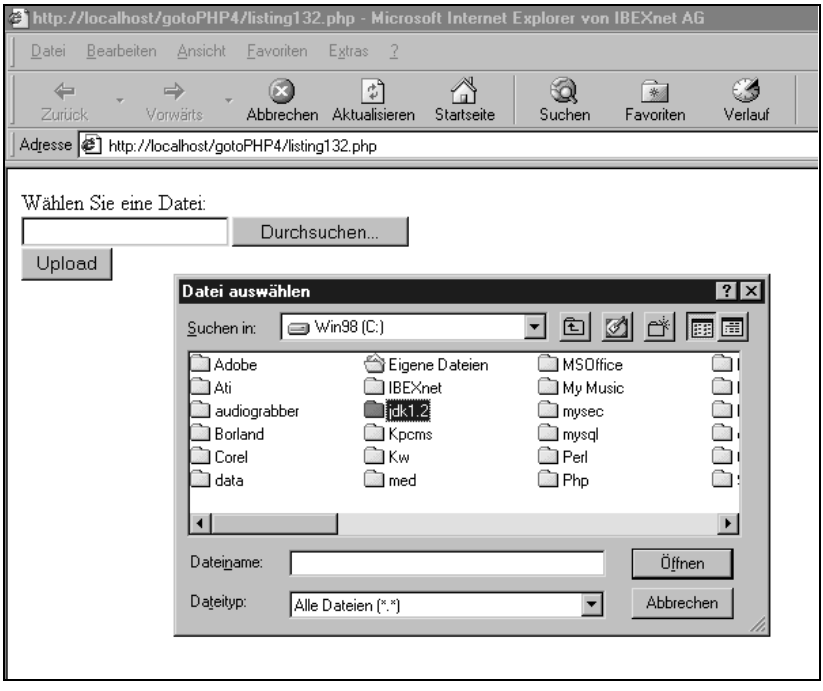


Abbildung 10.1: Dateiupload im Browser

Selbstverständlich können auch ganz reguläre Daten über das Formular verschickt werden. Sie werden wie gehabt als einfache Variablen oder Arrays übergeben.

10.6.2 Das PHP-Skript

Die Datei wird durch das HTML-Formular an das angegebene PHP-Skript übergeben und weiterverarbeitet. Genau wie bei regulären Formularen werden die Übergabedaten in Variablen zur Verfügung gestellt, mit denen das Skript zur Laufzeit arbeiten kann.

Nachdem das Formular abgeschickt worden ist, beginnt der Upload der Datei an den Zielserver. Die Daten werden durch den PHP-Interpreter entgegengenommen und als temporäre Datei im Ordner für Datei-Uploads gespeichert. Ist das geschehen, wird das eigentliche Zielskript gestartet, in dem vier Variablen definiert werden:

Variable	Wert
`$datei`	Der temporäre Name mit Pfad, unter dem die hochgeladene Datei auf dem Server gespeichert wurde
`$datei_name`	Der ursprüngliche Name der Datei auf dem Rechner des Users
`$datei_size`	Größe der Datei in Bytes
`$datei_type`	MIME-Type der Datei. Zum Beispiel image/gif

Tabelle 10.10: Informationen nach dem Dateiupload

Die Variable trägt den Namen `$datei`, da das Feld im Eingabeformular für den Dateiupload so genannt wurde. Diese Bezeichnung wird bei allen vier Variablen verwendet, lediglich der hintere Teil nach dem Unterstrich bleibt gleich.

Das PHP-Skript, dem die hochzuladende Datei übergeben wird, sollte alle nötigen Anweisungen enthalten, die festlegen, wie mit der Datei verfahren werden soll. Beispielsweise kann die `$file_size`-Variable verwendet werden, um Dateien auszusortieren, die zu groß oder zu klein sind. Oder man benutzt `$file_type`, um sich aller Dateien zu entledigen, die nicht bestimmten Typen entsprechen.

Da die temporäre Datei nach der Ausführung des Skripts automatisch aus dem Ordner gelöscht wird, muss sie zuvor in ein anderes Verzeichnis kopiert oder umbenannt werden. Das geht natürlich am einfachsten mit den Funktionen `copy()` oder `rename()`.

```
<FORM ENCTYPE="multipart/form-data" ACTION="listing132.php"
METHOD=POST>
Wählen Sie eine Datei:<br>
<INPUT NAME="datei" TYPE="file"><BR>
<INPUT TYPE="submit" VALUE="Upload">
</FORM>
<?
if (isset($datei))
{
        echo "Temporärer Dateiname: $datei<br>";
        echo "Original Dateiname: $datei_name<br>";
        echo "Größe der Datei in Byte: $datei_size<br>";
        echo "MIME-Type der Datei: $datei_type<br><br>";
        copy($datei, $datei_name);
        if($datei_type == "image/gif" or $datei_type == "image/
pjpeg")
```

```
        {
                echo "<img src='$datei_name' border='1'>";
        }
}
?>
```

Listing 10.16: Dateiupload an einen Server

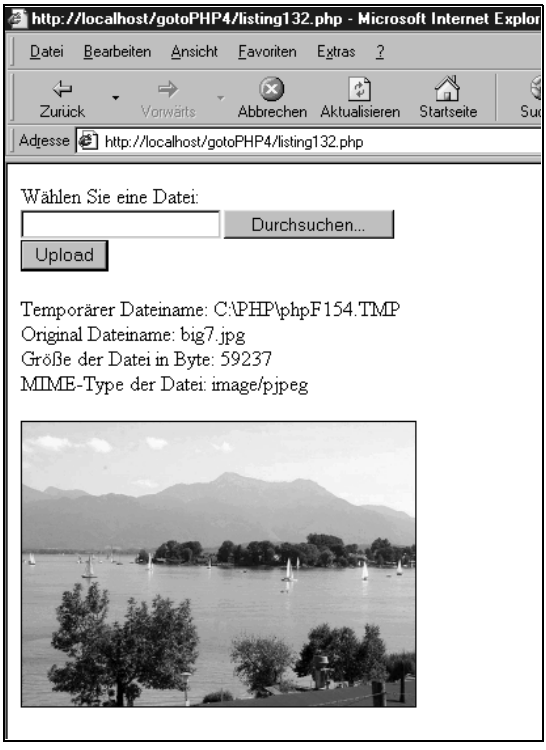


Abbildung 10.2: Upload eines Bildes

Das Skript, das die Datei nach dem Upload empfängt, wurde direkt nach dem HTML-Formular eingesetzt. Über die if-Bedingung am Anfang des Programms wird sichergestellt, dass das Skript nur ausgeführt wird, wenn auch tatsächlich eine Datei hochgeladen wurde.

In den ersten paar Zeilen werden alle Informationen über die empfangene Datei ausgegeben, die in den Dateivariablen gespeichert wurden. Dabei wird deutlich, dass der temporäre Dateiname `$datei` den kompletten Pfad zur neuen Datei im Übergangsverzeichnis speichert. Diese Eigenschaft

wird in der Funktion `copy()` genutzt, um die Daten vor dem Ende des Skripts in einem sicheren Verzeichnis abzulegen. Da als Ziel nur der ursprüngliche Name der Datei auf dem lokalen Rechner angegeben wurde, landet die Datei im aktuellen Skriptverzeichnis.

Im letzten Schritt des Programms wird überprüft, ob es sich bei der hochgeladenen Datei um ein Bild vom Typ GIF oder JPG handelt. Stimmt der MIME-Typ mit einem der beiden Formate überein, wird das Bild über den Browser ausgegeben. Die Belohnung ist ein entspannter Blick über den Chiemsee.

10.7 Übungen

Ü

1. Was ist der Unterschied zwischen den Zugriffsmodi w+ und a+ ?
2. Kann PHP auf Dateien anderer Server zugreifen?
3. Welchen Sinn macht ein Dateizeiger?
4. Was macht das Sonderzeichnen `\n` ?
5. Was gibt die Funktion `dir()` zurück?
6. Gibt es einen Verzeichniszeiger?

11 Datenbanken

Datenbanken gehören heute zur Standardausrüstung jedes Webservers und erlauben es, Webseiten mit aktuellen Informationen zu versorgen, ohne die Seite selbst verändern zu müssen. Der große Vorteil einer Datenbank ist die strikte Trennung zwischen dem Datenbestand und dem eigentlichen Programm, das für die Erstellung der Webseite genutzt wird.

PHP ist bekannt für den souveränen Umgang mit vielen unterschiedlichen Datenbanksystemen. Neben bekannten Programmen wie dBase, MySQL und Oracle unterstützt PHP auch eine ganze Reihe weiterer Softwarepakete:

- DBM-Datenbanken
- dBase
- Informix
- Microsoft SQL-Server
- mSQL
- MySQL
- ODBC (für Windows)
- Oracle 7/8
- PostgreSQL

Für viele der hier beschriebenen Datenbanken ist es nötig, nachträglich Treiberdateien zu installieren und den PHP-Interpreter neu zu kompilieren. Wenn Sie also mit einer bestimmten Datenbank arbeiten wollen, dann klären Sie mit Ihrem Systemadministrator, ob die vorhandene PHP-Version diese auch unterstützt.

Eine Ausnahme bildet die GNU-Distribution MySQL, die Sie kostenlos im Internet herunterladen können. Wegen ihrer überragenden Bedeutung im Zusammenhang mit PHP ist das MySQL-Modul seit der Version 4 automatisch mit installiert. MySQL gehört zu den am häufigsten genutzten Datenbanken im Internet und ist dank der GNU-Public License (genau wie der Apache-Server) kostenlos zu beziehen und auf fast jedem Server zu finden.

Wenn Sie PHP und MySQL selbst installieren möchten, dann lesen Sie dazu das Kapitel »Das Arbeitsumfeld« für weitere Informationen. Die folgenden Kapitel beziehen sich auf das Zusammenspiel zwischen PHP und der MySQL-Datenbank auf einem bestehenden Server.

11.1 Kapitelübersicht

- Was ist eine Datenbank?
- SQL und MySQL
- PHP und die MySQL-Datenbank
- Einige Tipps zum Schluss

11.2 Was ist eine Datenbank?

Einfach gesprochen ist eine Datenbank eine Zusammenstellung von unterschiedlichen Daten, die durch eine ordnende Struktur miteinander in Beziehung stehen. Das einfachste Beispiel für eine Datenbank ist das klassische Telefonbuch: Die hier verzeichneten Daten sind durch eine alphabetische Ordnung strukturiert und können durch die Beziehung der Datensätze Name und Nummer einfach genutzt werden. Das Telefonbuch ist so gesehen das simpelste Modell einer digitalen Datenbank in Druckform.

11.2.1 History

Die Ursprünge der modernen Datenbanksysteme liegen in den ausklingenden 60er Jahren des Computerzeitalters. Vor dieser Zeit wurden Daten, die für EDV-Systeme und Programme gespeichert werden mussten, in einfachen Dateiformaten abgelegt oder in programmabhängigen Datenstrukturen gespeichert.

Im Laufe der Zeit wurden die Datenmengen, die in einem System verwaltet wurden, immer größer, so dass eine Reihe von Problemen auftraten, die mit den bisherigen Methoden nicht gelöst werden konnten.

- Daten waren nur für bestimmte Programme lesbar, da es keine einheitliche Systematik gab.
- Mehrere Benutzer konnten nicht gleichzeitig auf dieselben Daten zugreifen, ohne sich zu stören.

- Daten waren system- und geräteabhängig.
- Anwender und Programmierer konnten nicht arbeiten, ohne die interne Struktur der Datenkonstrukte zu kennen.
- Daten wurden aufgrund unterschiedlicher Formate mehrfach gespeichert → Datenredundanz

Um diesen Problemen erfolgreich begegnen zu können, musste ein neues Datenverwaltungssystem entworfen werden, das unabhängig vom benutzten System und von der genutzten Software Daten auf Anfrage zur Verfügung stellt. Die Schnittstelle zwischen den Daten und der eigentlichen Software sollte möglichst neutral sein und für eine große Anzahl von Applikationen und Benutzern zu Verfügung stehen.

Die ersten Schritte in die richtige Richtung waren einfache Dateiverwaltungssysteme (SAM, ISAM), die Ende der 60er Jahre zum ersten Mal eingesetzt wurden. Gleichzeitig wurden die ersten Dienstprogramme zu dieser Software genutzt, die es unter anderem erlaubten, Daten zu filtern oder nach bestimmten Vorgaben zu sortieren.

In den 70er Jahren kamen dann die ersten Datenbanksysteme auf den Markt, die den modernen Ansprüchen an eine Datenbank genügten. Sie unterstützten die effiziente Verwaltung von großen Datenmengen und erlaubten den gleichzeitigen Zugriff von mehreren Benutzern. Die Datenunabhängigkeit wurde durch ein neuartiges 3-Ebenen-Konzept gewährleistet.

11.2.2 Aufbau und Konzept einer Datenbank

Der Zugriff auf die Daten einer Datenbank erfolgt in der Regel nicht direkt auf die Datenstruktur, sondern über ein so genanntes Datenbankmanagementsystem (DBMS). Dieses Softwaresystem wird auf eine bestehende Datenbank aufgesetzt und agiert als Schnittstelle zwischen den Daten und den zugreifenden Applikationen und Usern.

Über das DBMS ist es möglich, Daten einer Datenbank abzufragen, zu verändern oder komplett neu zu definieren. Meistens werden die Befehle an das DBMS über eine systemunabhängige Abfragesprache wie SQL formuliert, um eine standardisierte Schnittstelle zu bieten. Zusammen mit der Datenbank selbst ergibt das Datenbankmanagementsystem das so genannte Datenbanksystem.

Die Organisation eines Datenbanksystems gliedert sich in drei unterschiedliche Ebenen, die als externes, konzeptionelles und internes Schema bezeichnet werden. Der Unterschied zwischen diesen drei Ebenen besteht nicht im Aufbau der Datenbank, sondern vielmehr in den möglichen Sichtweisen einer bestehenden Datenstruktur.

Das externe Schema

Das externe Schema ist die äußere Sicht auf eine Datenbank. Je nach Benutzergruppe oder Applikation, die auf die Daten einer Datenbank zugreifen, ändert sich die externe Sicht auf die Daten, da unterschiedliche Interessensprofile vorliegen. Aus diesen Gründen gibt es auch niemals nur eine einzige externe Sicht auf eine Datenbank, sondern immer eine große Anzahl verschiedener Möglichkeiten.

Das konzeptionelle Schema

Während bei der externen Sicht, je nach Interessenslage, nur bestimmte Teile oder einzelne Datenkombinationen betrachtet werden, bezieht sich das konzeptionelle Schema auf alle Daten einer Datenbank. Es definiert den Aufbau und die Struktur einer Datenbank und legt fest, in welchen Beziehungen die einzelnen Datensätze stehen.

Das konzeptionelle Schema ist die Grundlage für den Einsatz eines Datenbankmanagementsystems, da es gewährleistet, dass die Daten einen sinnvollen Aufbau haben. Der Zugriff auf die Daten erfolgt immer über die konzeptionelle Ebene einer Datenbank. Im Gegensatz zur externen Sicht auf die Daten kann es von der konzeptionellen Ebene immer nur eine geben.

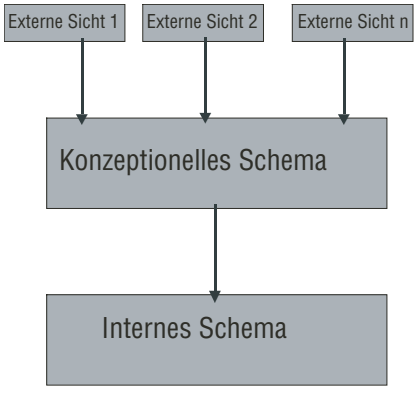


Abbildung 11.1: Die Ebenen einer Datenbank

Das interne Schema

Die interne Sicht beschreibt, wie die Daten einer Datenbank physisch auf einem Rechner angeordnet und gespeichert werden. Diese sehr rechnernahe Ebene bleibt den Benutzern einer Datenbank meistens verborgen, da sie durch die Oberfläche des DBMS überdeckt ist.

11.2.3 Relationale Datenbanken

Das relationale Datenbankmodell ist das bei weitem wichtigste Modell für den Aufbau einer Datenbank. Es beschreibt die Struktur einer Datenbank auf der konzeptionellen Ebene und schafft so eine Grundlage für ein DBMS. Dieses Modell wurde von E.F. Codd entwickelt und stellt Datenstrukturen anhand der realen Zusammenhänge in der Umwelt her. Grundlage für dieses Modell sind die neun Coddschen Regeln:

1. Integration durch einheitliche, nichtredundante Datenverwaltung
2. Operationen zum Speichern, Suchen und Ändern
3. Zugriff auf alle Datenbankbeschreibungen
4. Benutzersichten
5. Konsistenzüberwachung durch Integritätssicherung
6. Datenschutz durch Ausschluss unautorisierter Zugriffe
7. Zusammenfassung von DB-Operationen zu Funktionseinheiten
8. Synchronisation von parallelen Transaktionen
9. Datensicherung

Im relationalen Datenbankmodell werden die Daten in Form von Relationen erfasst, wie sie sich aus dem logischen Zusammenhang in der Realität ergeben. Das Ergebniss kann in Form einer Tabelle dargestellt werden, die alle Datensätze in einer zusammenhängenden Relation speichert.

Die Zusammenstellung von Namen und Telefonnummern einer Kontaktliste könnte als Relation grafisch so dargestellt werden:

Name	Telefonnummer
Albert Aal	12345678
Achim Armut	78957384

Name	Telefonnummer
Berta Allesschön	87548934
...	...

Der Relationsname wäre in diesem Fall der Name der Tabelle, beispielsweise Telefonnummern. Die Spalten der Tabelle (Name, Telefonnummer) sind die Attribute der Relation und beschreiben die Eigenschaften eines Datensatzes. Jede Relation kann durch ihre Attribute identifiziert werden. Die Kombination der Attribute nennt man Relationsschema. Die einzelnen Tupel der Relation bestehen aus den zusammenhängenden Datensätzen, die die eigentlichen Informationen der Datenbank speichern.

Eine Datenbank besteht in der Regel aus der Zusammenfassung verschiedener Relationen, die miteinander in Beziehung stehen. Eine sinnvolle Ergänzung zum oben beschriebenen Modell wäre beispielsweise eine Relation mit dem Namen Adressdaten, die neben Straße und Hausnummer auch den Ort mitsamt der Postleitzahl enthält. Würde man diese Daten mit einer weiteren Relation kombinieren, die Personalnummern und Gehalt und Einsatzort speichert, dann hätte man eine komplette Personaldatenbank mit allen nötigen Angaben.

Schlüssel

Jede Relation setzt sich aus einer endlichen Menge von Attributen zusammen, die die Eigenschaften der zu speichernden Datentupel bestimmen. Wenn ein Attribut oder eine Kombination von Attributen ein Datentupel eindeutig identifizieren kann, dann spricht man von einem Relationsschlüssel.

Im vorliegenden Beispiel der Relation Telefonnummern wäre das Attribut Telefonnummer der Schlüssel dieser Relation, da es ausgeschlossen ist, dass mehrere Personen dieselbe Nummer besitzen. Umgekehrt wäre es aber denkbar, dass verschiedene Personen denselben Namen tragen. Deshalb ist das Attribut Name nicht als Schlüssel geeignet.

Sind mehrere Attribute als Schlüssel geeignet, dann wird in der Regel ein Attribut bestimmt, das als Primärschlüssel gewertet wird.

Begriffe des Relationsmodells

Nachfolgend eine Liste mit alle Begriffen, die im Zusammenhang mit dem Relationsmodell von Datenbanken gefallen sind:

Begriff	Bedeutung
Attribut	Spalte einer Tabelle
Attributwert	Element eines Attributs
Relationsschema	Menge von Attributen
Relation	Menge von Zeilen einer Tabelle
Tupel	Zeile einer Tabelle
Datenbank	Menge von Relationen
Schlüssel	minimale Menge von Attributen, deren Werte einen Tupel einer Relation eindeutig identifizieren
Primärschlüssel	bestimmter Schlüssel einer Relation
Fremdschlüssel	Attributmenge, die in einer anderen Relation ein Schlüssel ist

Tabelle 11.1: Zusammenfassung aller Begriffe des Relationsmodells

MySQL ist eine relationale Datenbank. In dem folgenden Kapitel ist das Relationsmodell die Grundlage für alle Beispiele.

11.3 SQL und MySQL

Nachdem Sie auf den vorhergehenden Seiten einen kurzen Einblick in die Geschichte und die Hintergründe der Datenbanksysteme bekommen haben, können wir uns jetzt auf den praktischen Teil der Arbeit stürzen.

Wie ich im letzten Kapitel schon erwähnt habe, ist die Grundlage für die folgenden Erklärungen und Beispiele die MySQL-Datenbank, die zu den verbreitetsten Datenbanksystemen im Internet gehört. MySQL ist eine sehr mächtige Software, die bereits seit einigen Jahren erfolgreich weiterentwickelt wird und eine Menge interessanter Features zu bieten hat. Auch wenn Sie am Anfang die meisten dieser Techniken nicht brauchen, ist es ganz interessant zu wissen, wie diese Datenbank funktioniert.

Die folgende Übersicht zählt die wichtigsten Merkmal der MySQL-Datenbank auf:

- Multi-User-Zugriff möglich
- Schnittstelle für viele Programmiersprachen vorhanden, unter anderem C/C++, Java und natürlich PHP

- Für viele verschiedene Plattformen erhältlich
- optimierte SQL-Funktionen
- Kombination verschiedener Datenbanken in einer Abfrage ist möglich
- Komfortables Usermanagement
- ODBC-Unterstützung für Windows
- Kein Performanceverlust bei sehr großen Datenbanken (mehr als 50 Mio. Einträge)
- Unterstützung des europäischen Datensatzes ISO-8859-1 mit allen deutschen Sonderzeichen

Die Datenbank MySQL arbeitet mit der Abfragesprache SQL, die die Grundvoraussetzung für jeglichen Datenaustausch ist. Um die Kompatibilität zu anderen SQL-Datenbanken zu wahren, unterstützt MySQL zum großen Teil den ANSI SQL92-Standard, der den SQL-Dialekt definiert. Außnahmen bilden verschachtelte SELECT-Anweisungen und eine Reihe selten benutzter Features, die in der Praxis kaum Verwendung finden.

Neben den bekannten ANSI-Funktion bietet MySQL noch eine Reihe von Zusatzfunktionen, die den Umgang mit String und Daten vereinfachen. Unter anderem können Sie mit MySQL reguläre Ausdrücke und Stringvergleiche nutzen sowie auf eine eigene Datumsarithmetik zugreifen.

11.3.1 Einführung in SQL

Für die Arbeit mit MySQL ist die Kenntnis einiger grundsätzlicher Punkte notwendig, um erfolgreich mit der Sprache SQL umzugehen. Im Folgenden werde ich kurz die unterschiedlichen Schreibweisen und Konventionen dieser Sprache vorstellen. Im nächsten Kapitel werden Sie dann die Zugriffsbefehle von SQL kennen lernen.

Hinweis

Dieses Kapitel wird Ihnen in einer kurzen Einführung die Abfragesprache SQL näher bringen. Um die folgenden Beispiele nachvollziehen zu können, brauchen Sie ein MySQL-Interface, das es Ihnen erlaubt, die vorgestellten Befehle direkt einzugeben. Wenn Sie MySQL lokal installiert haben, können Sie das MySQL-eigene Zugriffsprogramm nutzen, das Sie unter folgendem Pfad auf Ihrem Rechner finden:

```
C:\mysql\bin\mysql.exe
```

Da das Programm keine graphische Oberfläche hat, öffnet sich ein DOS-Fenster, über das Sie alle Befehle eingeben können. Wie Sie die MySQL-Datenbank bei Ihnen lokal installieren, erfahren Sie im Kapitel »Das Arbeitsumfeld« am Ende des Buches.

Falls Sie direkt auf einem Server arbeiten, empfiehlt es sich, mit einem Administrationsprogramm zu arbeiten, das Ihnen direkten Zugriff auf die Datenbank gewährt.

Strings und Buchstabenketten

Strings werden in SQL, genau wie in PHP, in einfache oder doppelte Anführungszeichen gesetzt.

```
"Das ist eine Zeichenfolge!"
'Eine weitere Zeichenfolge!'
```

Die Möglichkeit, einfache Anführungszeichen zu verwenden, kommt PHP-Programmierern entgegen, weil Sie so Anfragestrings schnell und einfach zusammensetzen können. Mehr dazu in einem späteren Kapitel.

Zahlenwerte

Zahlen brauchen in SQL nicht in bestimmte Formatierungszeichen gesetzt zu werden. Integerwerte werden einfach als Folge einzelner Ziffern dargestellt. Kommazahlen müssen für die Trennung zwischen dem Ganzteil und dem Bruchteil einen Punkt (.) setzten. Ein Komma ist genau wie in PHP nicht gültig.

Negative Zahlen werden durch ein Minuszeichen gekennzeichnet.

```
12
3.1415927
-17
-23.5
```

Integerzahlen werden im Zusammenhang mit Kommazahlen ebenfalls als Kommazahlen interpretiert.

Der Ausdruck NULL

NULL bedeutet in SQL, dass keine Daten existieren. Dies ist nicht zu verwechseln mit 0 oder dem Nullstring »«, da diese Werte Daten darstellen.

Bezeichner

Genau wie Variablen oder Konstanten gelten für die Bezeichner von Datenbanken, Tabellen und Spalten gewisse Regeln. Der Name einer Datenbank darf maximal 64 Zeichen lang sein und kann nur aus Buchstaben und Zahlen bestehen. Das erste Zeichen muss ein Buchstabe sein, Leerzeichen sind nicht erlaubt. Tabellen und Spalten dürfen aus maximal 18 Zeichen bestehen. Ansonsten gelten für sie dieselben Regeln wie für Datenbanken.

Sollten diese Konventionen verletzt werden, wird das Datenbankmanagementsystem einen Fehler ausgeben, auch wenn die Syntax korrekt ist.

Die Datentypen von MySQL

Die Datenbank MySQL kennt eine Reihe unterschiedlicher Datentypen, die verschiedene Wertetypen darstellen können. Grundsätzlich kann man zwischen nummerischen Datentypen, Stringdatentypen und Datentypen für Zeitangaben unterscheiden.

In Tabelle 11.2 sind alle nummerischen Datentypen von MySQL sowie die möglichen Werte für diesen Speicherbereich zusammengefasst.

Typ	Beschreibung
`TINYINT`	Kleine Ganzahlen zwischen –128 und 127.
`UNSIGNED TINYINT`	Kleine Ganzahlen zwischen 0 und 255.
`SMALLINT`	Ganzzahlen zwischen –32768 und 32767.
`UNSIGNED SMALLINT`	Ganzzahlen zwischen 0 und 65535.
`MEDIUMINT`	Ganzzahlen zwischen –8388608 und 8388607.
`UNSIGNED MEDIUMINT`	Ganzzahlen zwischen 0 und 16777215.
`INT`	Ganzzahlen zwischen –2147483648 und 2147483647.
`UNSIGNED INT`	Ganzzahlen zwischen 0 und 4294967294.
`BIGINT`	Ganzzahlen zwischen –9223372036854775808 und 9223372036854775807.
`UNSIGNED BIGINT`	Ganzzahlen zwischen 0 und 18446744073709551615.
`FLOAT`	Kommazahlen
`DOUBLE`	Kommazahlen mit doppelter Genauigkeit oder doppelter Reichweite.

Tabelle 11.2: Nummerische Datentypen

Die nächste Tabelle enthält alle Datentypen in MySQL für den Umgang mit Zeichenketten.

Typ	Beschreibung
CHAR	Zeichenkette mit fester Länge, die bei der Initialisierung des Wertes festgelegt wird.
VARCHAR	Zeichenkette mit variabler Länge. Die maximale Länge wird bei der Initialisierung festgelegt.
TINYTEXT	Zeichenkette mit maximal 255 Zeichen.
TINYBLOB	Zeichenkette mit maximal 255 Zeichen.
BLOB	Zeichenkette mit maximal 65535 Zeichen.
TEXT	Zeichenkette mit maximal 65535 Zeichen.
MEDIUMTEXT	Zeichenkette mit maximal 16777215 Zeichen.
MEDIUMBLOB	Zeichenkette mit maximal 16777215 Zeichen.
LONGBLOB	Zeichenkette mit maximal 4294967295 Zeichen.
LONGTEXT	Zeichenkette mit maximal 4294967295 Zeichen.

Tabelle 11.3: String-Datentypen

Die Zeit- und Datumsdatentypen sind eine Besonderheit von MySQL und können nur hier verwendet werden. Sie erlauben es, verschiedene Zeitpunkte in unterschiedlichen Formaten »artgerecht« zu speichern.

Typ	Beschreibung
DATE	Datum: Wertebereich von 1000-01-01 bis 9999-12-31
DATETIME	Datum und Uhrzeit: Wertebereich von 1000-01-01 00:00:00 bis 9999-12-31 23:59:59
TIMESTAMP	UNIX Zeitstempel
TIME	Uhrzeit
YEAR	Jahr: zwei- oder vierstellig

Tabelle 11.4: Datentypen für Datum und Zeit

Die wichtigsten Datentypen und deren Anwendung werden im Kapitel über Tabellen genauer vorgestellt.

Mathematische Operatoren

MySQL verwendet alle geläufigen mathematischen Operatoren, die bereits aus anderen Programmiersprachen bekannt sind.

Operator	Beschreibung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

Tabelle 11.5: Mathematische Operatoren

Die Operatoren können problemlos in jeder SQL-Anfrage genutzt werden. Das folgende Beispiel zeigt den einfachsten aller anzunehmenden Fälle:

```
SELECT 5 + 4;
SELECT 5 * 4;
SELECT 5 - 4;
SELECT 5 / 4;
```

Ausgabe:

```
9
20
1
1.25
```

Divisionen durch 0 werden mit dem Ergebnis NULL gewertet.

Vergleichsoperatoren

Das Ergebnis von Kombinationen von Ausdrücken mit Vergleichsoperatoren ist entweder TRUE oder FALSE. Diese Angaben sind nötig, um bedingte Anfragen an eine Datenbank stellen zu können.

Operator	Beschreibung
=	Gleich
!=	Ungleich
<>	Ungleich
<=	Kleiner oder gleich
<	Kleiner
>=	Größer oder gleich
>	Größer

Tabelle 11.6: Vergleichsoperatoren

Im Gegensatz zu PHP oder anderen Programmiersprachen wird der Vergleich von zwei Werten in SQL mit einem einfachen Gleichheitszeichen realisiert. Der Operator ist also von einer Zuweisung nicht zu unterscheiden und muss aus dem Zusammenhang geschlossen werden.

Die Beispiele zeigen die Reaktion von SQL auf einfache logische Angaben:

```
SELECT 1 = 2;
SELECT 1 != 2;
SELECT 4  > 3;
SELECT 5 < 2;
```

Ausgabe:

```
0
1
1
0
```

TRUE und FALSE werden wie in PHP über die Ziffern 1 und 0 dargestellt. 0 steht für FALSE und 1 für TRUE.

Logische Operatoren

Logische Operatoren dienen zur Verknüpfung von Ausdrücken oder Bedingungen. Genau wie bei Vergleichsoperatoren wird entweder TRUE oder FALSE zurückgegeben.

Operator	Beschreibung
NOT	Negation
!	Negation, wie NOT
OR	Oder
||	Oder, wie OR
AND	und
&&	und, wie AND

Tabelle 11.7: Logische Operatoren

In den folgenden Kapiteln werden Sie der Reihe nach alle wichtigen Befehle und Möglichkeiten der MySQL-Software kennen lernen. Danach werden Sie ohne Probleme in der Lage sein eine Datenbank zu bedienen und mit PHP auf sie zuzugreifen.

Der besseren Übersicht wegen werde ich alle Befehle aus der MySQL Befehlsreferenz in Großbuchstaben angeben. Parameter oder Optionsangaben erfolgen in Kleinbuchstaben. In der Praxis ist diese Syntax allerdings nicht notwendig: MySQL akzeptiert die Befehle in jeder Schreibweise.

Wie Sie in den letzten Beispielen schon gesehen haben, werden SQL-Anweisungen immer mit einem Semikolon (;) abgeschlossen. Ein Befehl oder eine Anfrage ist nicht vollständig, wenn das Semikolon fehlt. In einem solchen Fall wird sie nicht ausgeführt.

11.3.2 Datenstrukturen erstellen und manipulieren

Eine Datenbanksoftware unterscheidet zwischen den Datenstrukturen und den eigentlichen Datensätzen, die in der Datenbank gespeichert werden sollen. Datenstrukturen dienen dazu, wie der Name schon sagt, die gespeicherten Daten in eine konkrete Struktur zu binden, die es erlaubt, auf sie zuzugreifen und sie zu verwalten.

In relationalen Datenbanken bestehen die Datenstrukturen aus den schon besprochen Relationen, die in Form von Tabellen vorliegen. Jede Relation muss Teil einer Datenbank sein, darum werden zusammengehörende Tabellen unter einem übergeordneten System (einer Datenbank) gespeichert. MySQL kann ohne Probleme eine große Anzahl solcher Datenbanken speichern, die nebeneinander existieren.

CREATE DATABASE

Mit dem Befehl `CREATE DATABASE` wird eine neue Datenbank erstellt. Als weitere Angabe erwartet MySQL den Namen, den die neue Datenbank haben soll.

```
CREATE DATABASE name;
```

Dieser Befehl erstellt die Datenbank »name« in der Datenbanksoftware. Sollte schon eine Datenbank existieren, die diesen Namen trägt, dann wird MySQL einen Fehler ausgeben.

Eine Datenbank wird im Betriebssystem in Form eines neuen Verzeichnisses dargestellt. Falls Sie die MySQL-Software lokal installiert haben, finden Sie das Datenverzeichnis unter:

```
c:\mysql\data\name
```

Falls Sie nicht das Standardverzeichnis bei der Installation gewählt haben, kann der Pfad abweichen.

USE

Der Befehl `USE` erlaubt es, die aktuelle Datenbank, die für alle Anfragen zugrunde liegt, zu wechseln. Ist die falsche oder gar keine Datenbank gewählt, wird die MySQL-Software die angefragten Daten nicht finden können und eine Fehlermeldung ausgeben.

```
USE name;
```

Mit diesem Befehl wird die Datenbank »name« ausgewählt und für alle folgenden Anfragen genutzt.

DROP DATABASE

Wenn eine Datenbank nicht mehr gebraucht wird, kann sie mit dem Befehl `DROP DATABASE` gelöscht werden. Als Parameter erwartet dieser Befehl den Namen der Zieldatenbank.

```
DROP DATABASE name;
```

Dieser Befehl sollte mit Vorsicht eingesetzt werden, da nicht nur die Datenbank gelöscht wird, sondern auch alle Daten, die innerhalb dieses Verzeichnisses gespeicht worden sind. MySQL löscht in diesem Fall ohne Rückfrage, darum ist es sinnvoll, vor dem Löschen sicherzustellen, dass man nichts Falsches aus der Datenbank entfernt.

CREATE TABLE

Mit dem Befehle `CREATE TABLE` wird in der aktuell ausgewählten Datenbank eine neue Relation, sprich eine neue Tabelle erstellt. Die Tabellendaten werden automatisch im Datenbankverzeichnis auf dem lokalen Rechner in drei verschiedenen Dateien gespeichert.

Nach dem Befehl `CREATE TABLE` erwartet MySQL den Namen der neuen Tabelle sowie einige Angaben zu den einzelnen Spalten, die diese Tabelle haben soll.

```
CREATE TABLE tabelle (...);
```

Der erste Teil der Befehlskette gleicht der Erstellung einer neuen Datenbank. Nach `CREATE TABLE` wird einfach der Name der neuen Relation angegeben, der in diesem Beispiel schlicht »tabelle« lautet. Danach folgen in runden Klammern weitere Angaben zu Spalten der Relation.

Jede Spalte besteht aus mehreren Informationen, die über verschiedene Parameter definiert werden:

- Name der Spalte: ein Bezeicher, über den die Daten in der Spalte identifiziert werden.
- Typ der Spalte: der Typ der Daten, die in dieser Spalte gespeichert werden können (Integerwerte, Strings etc.)
- Optionale Parameter: Schlüsselwörter, die das Verhalten dieser Spalte beeinflussen oder festlegen.

Diese Informationen werden in dieser Reihenfolge pro Spalte in den Klammern des Befehls als Parameter übergeben. Für jede Spalte müssen diese Angaben einzeln erfolgen und werden durch Kommata getrennt.

Der erste Parameter ist der einfachste, da er lediglich den Namen der Spalte festlegt. Achten Sie darauf, dass Sie keinen Namen mehrfach in einer Tabelle verwenden.

Der Typ einer Spalte wird durch die Angabe eines Datentyps definiert. Dabei kommen die Schlüsselwörter zum Einsatz, die im letzten Kapitel im Abschnitt »Die Datentypen von MySQL« vorgestellt worden sind. Die Angabe des Datentyps erfolgt immer hinter dem Namen der Spalte und ist nicht optional. Einige der Datentypen müssen durch weitere Parameter genauer definiert werden:

Datentypen	Beispiel	Beschreibung
TINYINT SMALLINT MEDIUMINT INT BIGINT	UNSIGNED INT(5)	Nach der Angabe des Typs muss in runden Klammern die maximale Anzahl der Stellen angegeben werden.
FLOAT DOUBLE	FLOAT(5,3)	Nach der Angabe des Typs muss die maximale Anzahl der Stellen sowie die maximale Anzahl der Nachkommastellen definiert werden.
CHAR	CHAR(25)	In runden Klammern wird die Anzahl der Zeichen festgelegt.
VARCHAR	VARCHAR(100)	In runden Klammern wird die maximale Anzahl der Zeichen festgelegt.

Tabelle 11.8: Parameter für Datentypen

Jede Spalte kann nur die Informationen speichern, die durch den Typ der Spalte erlaubt sind. Eine Integerspalte kann also keine Texte oder andere Daten außer Ganzzahlen speichern.

Die optionalen Parameter erlauben es, für bestimmte Spalten Eigenschaften festzulegen. MySQL unterstützt eine Reihe von Parametern, die in der folgenden Tabelle zusammengefasst werden.

Parameter	Beschreibung
AUTO_INCREMENT	Der Inhalt eines Feldes dieser Spalte vom Typ Integer wird nach dem Einfügen eines neuen Datensatzes automatisch um 1 erhöht. Es entsteht eine Nummerierung aller Datensätze. Die Spalte muss außerdem als Schlüssel definiert sein.
PRIMARY KEY	Diese Spalte wird als Primärschlüssel definiert. Jeder Wert darf nur noch einmal in dieser Spalte genutzt werden.
UNIQUE KEY	Diese Spalte wird als Schlüssel definiert. Wenn es keinen Primärschlüssel gibt, wird die erste Spalte eines UNIQUE_KEY zum Primärschlüssel.

Tabelle 11.9: Optionale Parameter

Diese Parameter werden an das Ende des Spaltendefinition gestellt, falls sie verwendet werden sollen.

In der folgenden SQL-Anweisung soll eine Tabelle definiert werden, die wir bereits im Kapitel über die Grundlagen von Datenbanken kennen gelernt haben. Es handelt sich um eine einfache Telefonliste mit Namen und Nummern, die jetzt um eine simple Nummerierung erweitert werden soll.

ID	Name	Telefonnummer
1	...	...
2	...	...
3	...	...

Tabelle 11.10: Tabelle für Namen und Telefonnummern

Die Tabelle enthält keine Daten, da diese erst später eingefügt werden sollen. In diesem Beispiel soll nur die Struktur der Relation geschaffen werden. Die SQL-Anweisung hat folgenden Aufbau:

```
CREATE TABLE telnum (
id INT(6) AUTO_INCREMENT UNIQUE KEY,
name VARCHAR(35),
telefon MEDIUMINT(18) PRIMARY KEY
);
```

Die Anweisung schafft eine neue Tabelle in der aktuellen Datenbank. Die Tabelle besteht aus drei Spalten, welche die Daten speichern. Die erste Spalte ist die ID des Datensatzes, die über den Parameter `AUTO_INCREMENT` automatisch gefüllt wird. Es werden Integerwerte bis maximal sechs Stellen zugelassen.

Die zweite Spalte speichert den Namen der Person in der Form des Typs VARCHAR. Es werden pro Name maximal 35 Zeichen zugelassen. Die letzte Spalte speichert die eigentliche Telefonnummer in der Relation und ist deshalb auch als 18-stellige MEDIUMINT-Spalte definiert. Da jede Telefonnummer nur einmal existieren kann, ist dieses Feld als Primärschlüssel definiert.

DROP TABLE

Der Befehl `DROP TABLE` hat eine ähnliche Aufgabe wie der Befehl `DROP DATABASE`. Er ermöglicht das Löschen von Tabellen in einer Datenbank mit allen Daten, die in dieser Tabelle gespeichert sind.

```
DROP TABLE telnum;
```

Der Aufruf würde die Tabelle »telnum« wieder entfernen. Es besteht die Möglichkeit, mehrere Tabellen gleichzeitig zu erstellen, indem die Tabellennamen durch Kommas voneinander getrennt aufgezählt werden.

OPTIMIZE TABLE

Der Befehl `OPTIMIZE TABLE` kann zur Optimierung nach vielen Löschvorgängen in einer Tabelle angewendet werden.

```
OPTIMIZE TABLE telnum;
```

Durch regelmäßige Pflege der Relationen in einer viel genutzten Datenbank kann die Zugriffsgeschwindigkeit erhöht werden.

ALTER TABLE

Mit dem Befehl `ALTER TABLE` kann der Aufbau oder die Struktur einer bestehenden Tabelle verändert werden. Der Aufbau dieses Befehls ist in den ersten Bestandteilen immer gleich strukturiert:

```
ALTER TABLE name ...
```

Nach dem einleitenden Befehl `ALTER TABLE` folgt der Name der Zieltabelle, die verändert werden soll. Danach stehen dem Programmierer verschiedene Möglichkeiten zur Verfügung, je nachdem wie die Tabelle verändert werden soll.

Die erste und gleichzeitig einfachste Anwendung von `ALTER TABLE` ist das Umbenennen der Zieltabelle. Hierfür wird das Schlüsselwort `RENAME` verwendet, das direkt nach dem Tabellennamen eingesetzt wird. Danach folgt der neue Name der Tabelle.

```
ALTER TABLE telnum RENAME telefonnummern;
```

Dieser Befehl ändert den Namen der Tabelle »telnum« in »telefonnummern« um.

Eine weitere sehr flexible Möglichkeit eine Tabelle zu verändern ist das Schlüsselwort `ADD`, das es erlaubt sowohl Eigenschaften als auch ganze Spalten in eine bestehende Tabelle zu integrieren. Je nachdem, was man ändern möchte, verwendet man eines der folgenden Schlüsselwörter mit Parametern:

- `ADD PRIMARY KEY` (name_der_spalte)
- `ADD UNIQUE` (name_der_spalte)

In diesen SQL-Befehlen wird die Tabelle »telnum« modifiziert und um weitere Schlüssel erweitert. Ob dieses Vorgehen sinnvoll ist, bleibt wohl fraglich.

```
ALTER TABLE telnum ADD PRIMARY KEY (telefon);
ALTER TABLE telnum ADD UNIQUE (name);
```

Wenn Sie eine neue Spalte in eine bestehende Tabelle einfügen wollen, müssen Sie ebenfalls ADD verwenden.

```
ALTER TABLE telnum ADD vorwahl SMALLINT(5);
```

Anstelle eines weiteren Schlüsselwortes folgt direkt nach ADD der Name der neuen Spalte mit der korrekten Typbezeichnung. Es kommt hier dieselbe Syntax zum Einsatz wie beim `CREATE TABLE`-Befehl, den Sie im letzten Abschnitt kennen gelernt haben. Das Beispiel erweitert die Tabelle »telnum« um die Spalte »vorwahl«, die Daten vom Typ `SMALLINT` speichert.

Mit dem optionalen Schlüsselwort `AFTER` ist es möglich, die genaue Position der neuen Spalte zu bestimmen. Über `AFTER` wird die Spalte angegeben, nach der die neue Spalte eingefügt werden soll.

```
ALTER TABLE telnum ADD vorwahl SMALLINT(5) AFTER name;
```

In diesem Beispiel wird die neue Spalte »vorwahl« direkt nach der Spalte »name« eingefügt, also vor der eigentlichen Rufnummer.

Wenn Sie eine einzelne Spalte direkt modifizieren wollen, dann steht Ihnen das Schlüsselwort `MODIFY` zur Verfügung, das es erlaubt, die Spaltendefinitionen zu überschreiben. Es ist also problemlos möglich, für eine Spalte neue Datentypen festzulegen:

```
ALTER TABLE telnum MODIFY vorwahl VARCHAR(100);
```

Diese Anweisung verändert den Datentyp der Spalte »vorwahl« von `SMALLINT(5)` in den Stringtyp `VARCHAR(100)`.

Das genaue Gegenteil von `ADD` ist das Schlüsselwort `DROP`, das es erlaubt, Bestandteile einer Tabelle zu löschen. Das gilt sowohl für Eigenschaften wie Primärschlüssel als auch für ganze Spalten.

```
ALTER TABLE telnum DROP vorwahl;
```

Die Anweisung löscht die Spalte »vorwahl« mit allen darin gespeicherten Daten.

11.3.3 Datensätze erstellen und manipulieren

Nachdem Sie nun wissen, wie Datenstrukturen mittels einfacher SQL-Befehle erstellt werden können, ist es jetzt Zeit für die Eingabe konkreter Datensätze. Es versteht sich von selbst, dass nur solche Daten in einer Datenbank gespeichert werden können, die durch vorher definierte Strukturen gestattet werden. Jeder Datensatz braucht also eine Tabelle, in der er gespeichert wird.

INSERT

Mithilfe des Befehls `INSERT` können Datensätze in eine Tabelle eingefügt werden. Die wichtigsten Formen des INSERT-Befehls werden mit dem Schlüsselwort `INTO` aufgebaut, um eine Tabelle direkt ansprechen zu können. Der Name der Zieltabelle wird nach `INTO` angegeben:

```
INSERT INTO telnum ...
```

Sie haben verschiedene Möglichkeiten den `INSERT`-Befehl zu nutzen, um Daten in eine Tabelle zu schreiben. Die einfachste Möglichkeit sieht vor, alle Daten direkt über das Schlüsselwort `VALUES` nach dem Tabellennamen zu übergeben. SQL verlangt in diesem Fall, dass die Reihenfolge der neuen Daten sich nach dem Aufbau der Tabelle richtet.

```
INSERT INTO telnum VALUES (0, "Dirk", 0123, 456789);
```

Dieser Befehl übergibt einen Datensatz, der in der Datenbank gespeichert werden soll, an die Tabelle »telnum«. Die erste Spalte bekommt den Wert 0 übergeben, da dieser Wert automatisch durch das DBMS generiert wird. Dank der Eigenschaft `AUTO_INCREMENT` wird die Spalte jeweils auf den Wert der Anzahl der Datensätze gesetzt.

In der Spalte »name« wird der Wert »Dirk« gespeichert, der als Zeichenkette übergeben werden muss. Da die Spalte als `VARCHAR` definiert ist, muss der Wert in Anführungszeichen gesetzt werden.

Die letzten beiden Werte sind einfache Integerzahlen, die direkt von der Datenbank verarbeitet werden können. Sie werden einfach als Zahlenwerte an die jeweilige Tabellenspalte übergeben. Alle Daten müssen sich nach der Struktur der Tabelle richten, da es ansonsten nicht möglich ist, die einzelnen Punkte einer Tabellenspalte zuzuordnen.

Eine etwas flexiblere Möglichkeit, Daten mit `INSERT` in eine Tabelle zu schreiben, ist die Möglichkeit alle Zielspalten im Befehl vorzugeben. So ist es beispielsweise möglich, nur Teile einer Tabelle zu füllen, ohne auf die

Struktur Rücksicht nehmen zu müssen. Die Spalten, die gefüllt werden sollen, werden nach dem Tabellennamen in Klammern, ähnlich wie die Werte durch Kommas getrennt, angegeben.

```
INSERT INTO telnum (vorwahl, telefon, name) VALUES (0987, 65432,
"Gaby");
```

Dieser Befehl trägt einen weiteren Datensatz in die Tabelle »telnum« ein, der die Telefonnummer von »Gaby« speichert. Im Gegensatz zum letzten `INSERT`-Befehl wurden nach dem Namen der Zieltabelle die Namen der Spalten angegeben, die gefüllt werden sollen.

Ihnen wird aufgefallen sein, dass diesmal komplett auf die Spalte »id« verzichtet wurde. Da der Wert automatisch generiert wird, ist es nicht nötig, ihn zu übergeben. Außerdem ist die Reihenfolge im Vergleich zum Tabellenaufbau verändert. Die Spalte »name« ist erst am Ende der Werte angegeben. Da die Reihenfolge aber durch die Informationen der ersten Klammern vorgegeben sind (die Namen der Zielspalten werden aufgezählt), ist diese Abweichung kein Problem.

Um Daten einzelnen Spalten direkt zuordnen zu können, kann das Schlüsselwort `SET` anstelle von `VALUES` verwendet werden. Es gibt Ihnen die Möglichkeit so genannte Name-Wert-Paare zu bilden, in denen der Spaltenname und der Wert durch Gleichheitszeichen einander zugeordnet werden.

```
INSERT INTO telnum SET name="Hans", vorwahl=0234, telefon=654321;
```

Dieser Befehl hat prinzipiell dieselbe Wirkung wie das letzte Beispiel, da er ebenfalls einen neuen Datensatz für die Tabelle »telnum« erstellt. Auch hier wird die Spalte »id« ausgelassen, da der Wert automatisch erstellt wird. Der Unterschied liegt diesmal bei der Syntax des Befehls. Nach SET werden einfach die Namen der Spalten und die Werte über Gleichheitszeichen zugeordnet.

REPLACE

`REPLACE` arbeitet genauso wie `INSERT`, mit dem Unterschied, dass kein neuer Eintrag erstellt wird, sondern ein schon bestehender Eintrag überschrieben wird. MySQL wählt automatisch den Datensatz, der überschrieben werden soll, der einen identischen Schlüssel in einer Spalte besitzt.

```
REPLACE INTO telnum VALUES (0, "Dirk", 0221, 404227);
```

Das Beispiel ersetzt den ersten Eintrag der Tabelle »telnum« durch einen neuen Datensatz mit einer neuen Vorwahl. Da die Telefonnummer als

identifizierender Schlüssel gleich geblieben ist, kann MySQL den Zieldatensatz eindeutig identifizieren.

Würde diese Eingabe mit dem INSERT-Befehl ausgeführt werden, wird MySQL dies mit einer Fehlermeldung quitieren.

```
INSERT INTO telnum VALUES (0, "Dirk", 0221, 404227);
```

Ausgabe:

```
Error 1062: Duplicate Entry ("404227" for Key 1)
```

Der Befehl REPLACE arbeitet mit exakt derselben Syntax wie der Befehl INSERT, der im letzten Abschnitt besprochen wurde. Es gelten alle Schlüsselwörter, die auch für INSERT gültig sind. Es muss lediglich INSERT durch REPLACE ersetzt werden.

UPDATE

Mit dem Befehl UPDATE können Werte von bestehenden Datensätzen manipuliert werden. Der Befehl orientiert sich von der Syntax her an der SET-Klausel, die Sie bereits beim INSERT und REPLACE-Befehl kennen gelernt haben.

```
UPDATE telnum SET ...
```

Nach SET werden durch Name-Wert-Paare, bestehend aus den Bezeichnern der Tabellenspalten und den zugeordneten Werten, die neuen Daten für die Zieltabelle angegeben.

```
UPDATE telnum SET name="Dirk", telefon=123456, vorwahl=01234;
```

Dieses Beispiel zeigt, wie die Werte »name«, »telefon« und »vorwahl« auf bestimmte Werte gesetzt werden. Es besteht kein Unterschied zur SET-Klausel des INSERT-Befehls, der Daten auf demselben Weg übergibt.

Der UPDATE-Befehl in der oben vorgestellten Form bezieht sich auf keinen bestimmten Datensatz, sondern betrifft die komplette Tabelle. Das bedeutet, dass alle Datensätze nach der Ausführung dieser Anweisung auf die Werte »Dirk«, 123456 und 01234 gesetzt wurden. Da dies wegen der Tabellendefinition (»telefon« ist ein Primärschlüssel) nicht möglich ist, wird MySQL einen Fehler ausgeben.

Um den Zugriff auf bestimmte Datensätze zu fokussieren, können mit dem Schlüsselwort WHERE Bedingungen spezifiziert werden.

```
UPDATE telnum SET name="Dirk", telefon=123456, vorwahl=01234 WHERE
...
```

Hinter `WHERE` wird eine einfache Bedingung angegeben, die entweder TRUE oder FALSE zurückgibt. Diese Bedingung wird für jeden Datensatz kontrolliert, bevor dieser durch den `UPDATE`-Befehl manipuliert wird. Ist die Bedingung FALSE, wird der Datensatz nicht verändert.

```
UPDATE telnum SET name="Dirk", telefon=123456, vorwahl=01234 WHERE
id = 2;
```

Das Beispiel zeigt, wie auf den Datensatz mit der ID-Nummer 2 zugegriffen wird, da er der einzige Datensatz ist, auf den die angegebene Bedingung zutrifft. Es ist selbstverständlich möglich, mehrere Bedingungen durch logische Operatoren zu verknüpfen. Alle nötigen Schlüsselwörter haben Sie in der Einführung zu MySQL kennen gelernt.

```
UPDATE telnum SET name="Dirk", telefon=123456, vorwahl=01234 WHERE
id < 2 OR id > 5;
```

Der Befehl demonstriert, wie durch mehrere Bedingungen ein bestimmter Bereich von Datensätzen definiert werden kann. Die Beispielanweisung betrifft alle Datensätze bis auf die Daten mit der ID-Nummer 2 bis 5.

DELETE FROM

Mit dem Befehl `DELETE FROM` können Daten in einer Tabelle gelöscht werden. Nach `FROM` wird die Zieltabelle angegeben, in der die Datensätze gespeichert sind.

```
DELETE FROM telnum;
```

Werden keine weiteren Angaben gemacht, bezieht sich diese Anweisung auf die ganze Tabelle und alle Datensätze werden gelöscht. Mit Hilfe der `WHERE`-Klausel können Datensätze spezifiziert werden, so wie Sie es schon im letzten Abschnitt kennen gelernt haben.

```
DELETE FROM telnum WHERE id > 4;
```

Die Anweisung löscht alle Datensätze, deren ID-Nummer höher als 4 ist.

11.3.4 Datensätze mit SELECT auslesen

Der Befehl `SELECT` ist das wichtigste Werkzeug für die Abfrage von Daten aus einer relationalen Datenbank. Er dient dazu, Informationen aus einer oder mehreren Tabellen einer Datenbank zu erhalten und diese abhängig von bestimmten Merkmalen zu präsentieren.

Der SELECT-Befehl besteht aus mehreren Teilen: Der erste Teil wird durch das Schlüsselwort SELECT eingeleitet und definiert die Spalten der Zieltabellen, die ausgelesen werden sollen. SQL übernimmt dabei einfach die Spaltennamen, wie sie bei der Tabellendefinition eingerichtet wurden.

```
SELECT name, vorwahl, telefon ...
```

Die Spaltennamen werden einfach durch Kommas voneinander getrennt. Sollen alle verfügbaren Spalten einer Tabellenrelation auf einmal ausgelesen werden, dann können die Spaltennamen durch die Wildcard * ersetzt werden.

```
SELECT * ...
```

Das Sternchen symbolisiert, dass alle Daten, die dieser Befehl erreicht, in der so genannten Ergebnisrelation zurückgegeben werden sollen.

Der zweite Teil des SELECT-Befehls bestimmt den Namen der Zielrelationen, aus denen die Daten ausgelesen werden sollen. Dieser Teil wird durch das Schlüsselwort FROM eingeleitet und schließt direkt nach dem Spaltennamen an den Befehl an.

```
SELECT name, vorwahl, telefon FROM telnum;
```

Dieser Befehl liest alle Datensätze der Tabelle »telnum« aus, die in den Spalten »name«, »vorwahl« und »telefon« gespeichert sind. Das Ergebnis wird in einer neuen Ergebnistabelle ausgegeben, die alle Datensätze präsentiert.

Für die folgenden Beispiele gehe ich davon aus, dass die Tabelle »telnum« mit den folgenden SQL-Befehl erstellt und mit Daten gefüllt wurde. Da die letzten Beispiele sehr viel mit »telnum« experimentiert haben, ist es notwendig, ein wenig Klarheit über den Zustand der Tabelle zu gewinnen.

```
CREATE TABLE telnum (
id INT(6) AUTO_INCREMENT UNIQUE KEY,
name VARCHAR(35),
vorwahl MEDIUMINT(6),
telefon MEDIUMINT(10) PRIMARY KEY);
INSERT INTO telnum VALUES (0, "Dirk", 01234, 654321);
INSERT INTO telnum VALUES (0, "Gaby", 03432, 343331);
INSERT INTO telnum VALUES (0, "Hans", 08432, 1654);
INSERT INTO telnum VALUES (0, "Bert", 034, 6321);
INSERT INTO telnum VALUES (0, "June", 087, 123456);
```

Listing 11.1: Die Tabelle telnum

Die Relation hat also fünf Datensätze, die jeweils den Namen, die Vorwahl und die Rufnummer einer fiktiven Person darstellen. Zusätzlich sind alle Einträge durch eine fortlaufende Nummer gekennzeichnet.

Das erste Beispiel für den `SELECT`-Befehl gibt also folgendes Ergebnis zurück:

name	vorwahl	telefon
Dirk	1234	654321
Gaby	3432	343331
Hans	8432	1654
Bert	34	6321
June	87	123456

Tabelle 11.11: Die Ergebnisrelation

Die Ergebnisrelationen für eine SQL-Anfrage werden ebenfalls in Tabellenform zurückgegeben. Es fällt auf, dass die MySQL-Datenbank allen Vorwahlnummern die führende Null abgeschnitten hat. Das liegt daran, dass die Tabellenspalte »vorwahl« als Integerwert angegelegt worden ist und deshalb nur Zahlenwerte speichern kann. MySQL akzeptiert keine Zahlen, die mit einer Null beginnen, außer es handelt sich um die einzige Vorkommastelle einer Kommazahl.

Die WHERE-Klausel

Ähnlich wie beim `UPDATE`-Befehl kann eine `SELECT`-Anfrage mit der `WHERE`-Klausel eingeschränkt werden. Das Schlüsselwort `WHERE` wird direkt nach der Angabe der Zieltabelle eingesetzt und erlaubt es, Bedingungen an die Datensätze zu stellen.

```
SELECT name, vorwahl, telefon FROM telnum WHERE id = 2;
```

Diese einfache Anfrage schränkt die Ergebnisrelation auf alle Datensätze mit der ID-Nummer gleich 1 ein. In Tabelle »telnum« ist das lediglich ein Datensatz, darum sieht das Ergebnis so aus:

name	vorwahl	telefon
Gaby	3432	343331

Tabelle 11.12: Ergebnisrelation mit WHERE-Klausel

Die WHERE-Klausel kann in jeder möglichen Kombination von logischen Verknüpfungen und Bedingungen eingesetzt werden, welche die Operatoren aus dem vorhergehenden Kapitel erlauben.

Es ist durchaus möglich, auch komplett unsinnige Bedingungen zu stellen, die nach den bestehenden Strukturvorgaben niemals TRUE sein können.

```
SELECT name, vorwahl, telefon FROM telnum WHERE vorwahl = "Düssel-
dorf" and name > 124;
```

MySQL kontrolliert nur die Syntax einer Anfrage, nicht aber die semantischen Zusammenhänge mit der Zielrelation. Die Beispielanfrage würde schlicht und einfach eine leere Ergebnistabelle zurückgeben.

Mehrere Tabelle gleichzeitig auslesen

SQL ermöglicht es, ohne Probleme mehrere Tabellen mit dem SELECT-Befehl gleichzeitig auszulesen. Es ist sogar möglich, Tabellen unterschiedlicher Datenbanken gleichzeitig zu erfassen, ohne zwischen den Datenbanken wechseln zu müssen. Das Ergebnis wird einfach in einer großen Tabelle zusammengefasst und ausgegeben.

Um Überschneidungen bei der Abfrage zu vermeiden, falls mehrere Tabellen oder Spalten den gleichen Namen haben, kann jede Spalte und jede Tabelle in SQL mit einem eindeutigen Pfad identifiziert werden. Dieser Pfad setzt sich einfach aus den Namen der Datenbank, dem Namen der Tabelle und den Namen der Spalte durch Punkte getrennt zusammen.

Pfad	Beschreibung
name.telnum.telefon	Die Spalte »telefon« wird in einer eindeutigen Angabe über den Namen der Datenbank und der Tabelle identifiziert.
telnum.telefon	Die Spalte »telefon« wird innerhalb einer festgelegten Datenbank eindeutig identifiziert.
name.telnum	Die Tabelle »telnum« wird eindeutig über die Datenbank identifiziert.

Tabelle 11.13: Pfadangaben in SQL

Je nach Situation und Notwendigkeit kann der Pfad teilweise oder ganz angegeben werden. Ist die Adressierung ohnehin eindeutig, kann auf die globale Adressierung mit der Punktnotation verzichtet werden. SQL findet automatisch die zusammenhängenden Relationen.

Um mehrere Tabellen gleichzeitig auslesen zu können, ist es erst einmal notwendig, dass mehrere Tabellen existieren. Für das folgende Beispiel gehe ich davon aus, dass eine weitere Tabelle mit dem Namen »geburtstag« existiert.

```
CREATE TABLE geburtstag (name VARCHAR(32), datum DATE);
INSERT INTO geburtstag VALUES ("Dirk", "1978-11-30");
INSERT INTO geburtstag VALUES ("Gaby", "1982-1-30");
INSERT INTO geburtstag VALUES ("Opa", "1935-4-4");
INSERT INTO geburtstag VALUES ("Oma", "1942-7-12");
INSERT INTO geburtstag VALUES ("Hans", "1964-7-15");
INSERT INTO geburtstag VALUES ("June", "1972-8-19");
```

Listing 11.2: Die Tabelle »geburtstag«

Die einfachste Möglichkeit, beide Tabellen gleichzeitig auszulesen, besteht darin, die Tabellennamen nach FROM durch Kommas getrennt anzugeben.

```
SELECT * FROM geburtstag, telnum;
```

MySQL gibt in diesem Fall eine Tabelle mit allen Werten der beiden Relationen zurück, die in allen möglichen Kombinationen miteinander verknüpft sind. Die Ergebnismenge umfasst also 6 * 5 = 30 Datensätze, die in folgendem Aufbau dargestellt werden.

name	datum	id	name	vorwahl	telefon
...	...	...	...	...	...

Das Ergebnis ist zweifelsohne nicht optimal, da die Datensätze in einer sehr unübersichtlichen Struktur miteinander verknüpft worden sind. In der Praxis begegnet man diesem Problem durch verknüpfende `WHERE`-Klauseln, die dafür sorgen, dass nur die interessanten Möglichkeiten der Tabellenkombination ausgegeben werden.

```
SELECT  telnum.name, datum, vorwahl, telefon FROM geburtstag, telnum
WHERE geburtstag.name = telnum.name;
```

Diese Anweisung gibt nur die Datensätze aus, die sowohl in der Tabelle »telnum« als auch in der Tabelle »geburtstag« in der Spalte »name« denselben Wert haben. Das Ergebnis ist eine Relation, die für jeden Namen den richtigen Geburtstagstermin sowie die Telefonnummer mit Vorwahl ausgibt. Die Datensätze, die keine Übereinstimmung zeigen, werden stillschweigend ignoriert.

Ein weiterer interessanter Punkt an diesem Beispiel ist der Umstand, dass die Spalte »name« in der Anfrage über den Tabellennamen angegeben werden muss. Würde man nur den Spaltennamen übergeben, würde MySQL sich beschweren, da »name« sowohl in der Tabelle »geburtstag« als auch in »telnum« vorkommt.

Weitere Möglichkeiten mit SELECT

SQL bietet noch ein paar weitere recht nützliche Möglichkeiten, wie mit dem `SELECT`-Befehl Datensätze aus einer Datenbank aufbereitet werden können. Die erste und einfachste Möglichkeit, den Aufbau von Ergebnisrelationen zu verändern, ist das Schlüsselwort `ORDER BY`, das alle Datensätze in eine sortierte Ordnung gliedert.

`ORDER BY` wird immer an das Ende einer Anfrage gestellt und erwartet die Angabe einer Spalte, nach der die Datensätze sortiert werden sollen. Wenn keine weiteren Parameter angegeben werden, dann werden die Daten immer alphabetisch oder nummerisch (je nach Datentyp) sortiert und zurückgegeben.

```
SELECT * FROM telnum ORDER BY name;
```

Diese Anweisung gibt alle Daten der Tabelle »telnum« zurück. Im Gegensatz zu den anderen Beispielen wird die Ergebnisrelation anhand der Spalte »telnum« alphabetisch sortiert.

id	name	vorwahl	telefon
4	Bert	34	6321
1	Dirk	1234	654321
2	Gaby	3432	343331
3	Hans	8432	1654
5	June	87	123456

Tabelle 11.14: Ergebnisrelation mit ORDER BY

Mithilfe der Parameter `ASC` und `DESC` kann die Reihenfolge der Sortierung beeinflusst werden, indem der jeweilige Parameter nach der `ORDER BY`-Anweisung angegeben wird. MIT `ASC` wird eine aufsteigende Reihenfolge erzwungen und mit `DESC` eine absteigende Reihenfolge.

```
SELECT * FROM telnum ORDER BY name DESC;
```

Dieselbe Anweisung wird wie `DESC` umgedreht, so dass die Ergebnisrelation eine umgekehrte Reihenfolge aufweist.

id	name	vorwahl	telefon
5	June	87	123456
3	Hans	8432	1654
2	Gaby	3432	343331
1	Dirk	1234	654321
4	Bert	34	6321

Tabelle 11.15: Ergebnisrelation mit ORDER BY … DESC

Das Schlüsselwort `DISTINCT` filtert alle Datensätze aus dem Ergebnis heraus, welche exakt die identischen Werte aufweisen. Es wird jeweils nur ein Datensatz der Mehrfacheinträge angezeigt. Ohne `DISTINCT` würden wiederholte Einträge genauso oft angezeigt werden, wie sie in der jeweiligen Tabelle auftreten.

`DISTINCT` wird einfach nach `SELECT` in die Anweisung eingebaut.

```
SELECT DISTINCT * FROM telnum;
```

11.3.5 Strukturinformationen abfragen

Mithilfe von verschiedenen SQL-Befehlen können Informationen über Tabellen, Datenbanken, Spalten oder den MySQL-Server abgefragt werden.

SHOW

Der Befehl `SHOW` ist äußerst flexibel einsetzbar und ermöglicht es Informationen über die verschiedenen Bestandteile einer Datenbank zu erhalten. Je nachdem welches Schlüsselwort `SHOW` übergeben wird, werden andere Bereiche der Datenbank analysiert.

Um alle verfügbaren Datenbanken auf dem Server zu betrachten, wird das Schlüsselwort `DATABASES` angegeben.

```
SHOW DATABASES;
```

Das Ergebnis ist eine einspaltige Tabelle, in der alle Datenbanken aufgezählt werden.

Databases
mysql
name
test
home

Tabelle 11.16: Übersicht über die verfügbaren Datenbanken

Neben der bekannten Datenbank »name« stehen noch drei weitere Datenbanken zur Verfügung, die zum Teil anderen Projekten angehören. Die Datenbank »mysql« gehört zur MySQL-Software. Hier werden alle Daten zur Userverwaltung gespeichert.

Der nächste Schritt ist eine komplette Übersicht über alle Tabellen einer bestimmten Datenbank. `SHOW` macht dies über das Schlüsselwort `TABLES` möglich, wobei die Zieldatenbank über eine `FROM`-Klausel angegeben wird.

```
SHOW TABLES FROM name;
```

Auch hier wird das Ergebnis in einer einspaltigen Tabelle ausgegeben.

Tables
geburtstag
telnum

Tabelle 11.17: Übersicht über alle Tabellen der Datenbank »name«

Um nun Informationen über die Spalten einer bestimmten Tabelle zu erhalten, müssen Sie noch weiter ins Detail gehen. Hier kommt das Schlüsselwort `COLUMNS` zum Einsatz:

```
SHOW COLUMNS FROM telnum;
```

Das Ergebnis ist eine sechsspaltige Relation, die pro Spalte der Zieltabelle einen Datensatz zurückgibt.

Field	Type	Null	Key	Default	Extra
id	int(6)		UNI	<-NULL->	auto_increment
name	varchar(35)	YES		<-NULL->	
vorwahl	mediumint(6)	YES		<-NULL->	
telefon	mediumint(10)		PRI	0	

Tabelle 11.18: Informationen über die Tabelle »telnum«

Die erste Spalte gibt Auskunft über den Spaltennamen der Zieltabelle. Hier wird der Bezeichner angegeben, der bei der Erstellung der Tabelle übergeben wurde. Die zweite Spalte zeigt den Datentyp, der in der jeweiligen Spalte gespeichert werden kann. Die dritte Spalte ist ein Eigenschaftsfeld, das festhält, ob diese Spalte Null-Werte, also keine Daten speichern darf. Dies ist eigentlich immer der Fall, außer es handelt sich um Schlüsselfelder.

Die Spalte »Key« gibt an, ob es sich um eine Schlüsselspalte handelt. Wenn ja, wird hier der Typ des Schlüssels angegeben. Die fünfte Spalte speichert die Standardwerte für die jeweilige Spalte der Zielrelation. In der Spalte »Extras« werden eventuelle optionale Informationen gespeichert, wie beispielsweise die Eigenschaft »auto_increment«.

Zu guter Letzt möchte ich noch das Schlüsselwort `STATUS` vorstellen, das in Verbindung mit `SHOW` eine Übersicht über den aktuellen Status der Datenbank liefert.

```
SHOW STATUS;
```

Das Ergebnis ist eine zweispaltige Tabelle, die eine Vielzahl von Werten bezüglich der Datenbank zurückgibt.

DESCRIBE

Der Befehl `DESCRIBE` hat eine ähnliche Funktion wie `SHOW`, da er Informationen über Tabellen zurückgibt. Der Unterschied liegt im Detail: `DESCRIBE` arbeitet nur innerhalb einer Datenbank und kann deshalb keinen Überblick über die MySQL-Software geben.

Je nachdem, welche Informationen von DESCRIBE zurückgegeben werden sollen, erwartet der Befehl einen oder zwei Parameter. Der erste Parameter ist immer ein Tabellenname innerhalb der aktiven Datenbank.

Alternativ kann auch über die Punktnotation eine Tabelle einer anderen Datenbank ausgewählt werden.

```
DESCRIBE name.telnum;
```

Das Ergebnis dieser Anfrage ist dasselbe wie bei dem Befehl `SHOW TABLES`. Es wird eine Tabelle mit allen relevanten Informationen über die Zieltabelle erstellt.

Möchte Sie spezielle Informationen über eine einzelne Spalte, kann über einen zweiten optionalen Parameter der Name der Zielspalte angegeben werden.

```
DESCRIBE name.telnum telefon;
```

Diese Anfrage gibt ebenfalls eine sechsspaltige Tabelle zurück, die komplett dem Ergebnis der ersten Anfrage gleicht. Der Unterschied liegt in der Anzahl der Datensätze, da jetzt nur eine Spalte betrachtet wird.

Field	Type	Null	Key	Default	Extra
telefon	mediumint(10)		PRI	0	

Tabelle 11.19: Die Spalte »telefon«

Der Befehl `DESCRIBE` kann durch `DESC` abgekürzt werden.

11.3.6 Spezielle Funktionen von MySQL

MySQL bietet eine große Menge von integrierten Funktionen, die auf Abfrageergebnisse direkt angewendet werden können. Es wird zwischen vier verschiedenen Arten von MySQL-Funktionen unterschieden: Die mathematischen Funktionen, die Stringfunktionen, Datums- und Zeitfunktionen und die so genannten Aggregatsfunktionen, die für spaltenorientiertes Arbeiten eingesetzt werden.

Die hier vorgestellten Funktionen haben noch nichts mit PHP zu tun, auch wenn es teilweise aufgrund der Ähnlichkeit so wirken mag. Alle folgenden Funktion werden innerhalb von SQL-Anfragen eingesetzt, so dass das Ergebnis direkt in der Ergebnisrelation erscheint.

Mathematische Funktionen

MySQL unterstützt eine ganze Reihe von mathematischen Funktionen, die sowohl in Sachen Geschwindigkeit als auch Genauigkeit den PHP-Funktionen in nichts nachstehen. Oft ist es sogar einfacher, eine Berechnung direkt in MySQL vorzunehmen, bevor die Daten aus der Datenbank in das PHP-Programm übernommen werden.

Alle folgenden Funktionen können nur auf nummerische Datenspalten angewendet werden. Die Formel wird direkt in den SQL-Befehl eingebunden.

Funktion	Beschreibung
`ABS(n)`	Gibt den Absolutwert zurück
`MOD(n,m)`	Gibt den Rest einer Ganzzahlendivision zurück (n / m)
`FLOOR(n)`	Rundet den Wert auf
`CEILING(n)`	Rundet den Wert ab
`ROUND(n,m)`	Rundet den Wert mathematisch korrekt. Wenn der optionale Parameter m angegeben ist, wird auf m Nachkommastellen gerundet.
`EXP(x)`	Gibt die Potenz des Arguments zur Basis e (Eulersche Zahl) zurück
`LOG(x)`	Natürlicher Logarithmus
`LOG10(x)`	Logarithmus zu 10
`POW(x,y)`	Potenziert x mit y
`SQRT(x)`	Zieht die zweite Wurzel aus x
`PI()`	Gibt die Kreiszahl pi zurück (3.1415...)
`SIN(x)`	Sinusfunktion
`COS(x)`	Cosinusfunktion
`TAN(x)`	Tangensfunktion
`RAND()`	Gibt eine Zufallszahl zwischen 0 und 1 zurück
`LEAST(x, y, z, ...)`	Gibt den kleinsten Wert einer Liste zurück
`GREATEST(x, y, z, ...)`	Gibt den größten Wert einer Liste zurück
`TRUNCATE(x, y)`	Schneidet eine Dezimalzahl auf y Nachkommastellen ab

Tabelle 11.20: Mathematische Funktionen von MySQL

Alle Funktionen können direkt auf die Spaltennamen einer Tabelle angewendet werden. MySQL setzt automatisch die ausgelesenen Werte an die Stelle der Platzhalter und gibt das korrekte Ergebnis aus.

Für die folgenden Beispiele gehe ich von einer weiteren Tabelle mit dieser Form aus:

```
CREATE TABLE test (string VARCHAR(20), zahl float(5, 4), datum DATE);
INSERT INTO test VALUES ("Hallo", 127.175, "2001-12-24");
INSERT INTO test VALUES ("Wolkenbruch", 10143, "1854-4-13");
INSERT INTO test VALUES ("Das ist ein String", -2469.5, "1999-11-
30");
```

Listing 11.3: Die Tabelle »test«

Die Tabelle stellt jetzt eine Reihe von Werten zur Verfügung, die für die folgenden Beispiele als Vorlage dienen.

```
SELECT ROUND(zahl) FROM test;
```

Ausgabe:

```
127
10143
-2469
```

```
SELECT ABS(zahl) FROM test WHERE string = "Das ist ein String";
```

Ausgabe:

```
2469.000
```

```
SELECT TRUNCATE((PI() * zahl) / RAND(), 2) FROM test;
```

Ausgabe:

```
867.69
50668.54
-10005.01
```

```
SELECT SIN(PI());
```

Ausgabe:

```
0.000000
```

Stringfunktionen

Neben den mathematischen Funktionen bietet MySQL eine sehr große Anzahl von Stringfunktionen, die auf alle Datenbankfelder mit dem Datentyp `CHAR` oder `VARCHAR` angewendet werden können. Folgende Funktionen stehen zur Verfügung:

Funktion	Beschreibung
`ASCII(str)`	Gibt den ASCII-Code des ersten Zeichens des Strings zurück.
`CHAR(int)`	Wandelt den Integerwert in einen Buchstaben um.
`CONCAT(str1, str2, ...)`	Verknüpft alle Argumente zu einem Gesamtstring.
`LENGTH(str)`	Gibt die Anzahl der Zeichen zurück.
`POSITION(substr, str)`	Gibt das erste Auftreten des Teilstring substr im String str zurück.
`LPAD(str, len, str2)`	Fügt den String str2 links an den String str an. Die Funktion gibt len Zeichen zurück.
`RPAD(str, len, str2)`	Fügt den String str2 rechts an den String str an. Die Funktion gibt len Zeichen zurück.
`LEFT(str, len)`	Gibt len Zeichen von der linken Seite des Strings str zurück.
`RIGHT(str, len)`	Gibt len Zeichen von der rechten Seite des Strings str zurück.
`SUBSTRING(str, pos, len)`	Gibt len Zeichen des Strings str zurück, beginnend bei der Position pos.
`LTRIM(str)`	Entfernt führende Leerzeichen
`RTRIM(str)`	Entfernt Leerzeichen vom Ende des Strings.
`SPACE(len)`	Gibt einen String mit len Leerzeichen zurück.
`REPLACE(str, r1, r2)`	Ersetzt im String str alle Vorkommen von r1 durch r2.
`REPEAT(str, num)`	Wiederholt den String str num-mal.
`REVERSE(str)`	Dreht den String str herum.
`INSERT(str, pos, len, str2)`	Überschreibt den String str ab der Position pos bis zum Zeichen len mit dem String str2.

Tabelle 11.21: Stringfunktionen von MySQL

Funktion	Beschreibung
`ELT(n, str1, str2, str3, ...)`	Gibt den n-ten String aus der Liste zurück.
`LOWER(str)`	Wandelt in Kleinbuchstaben um.
`UPPER(str)`	Wandelt in Großbuchstaben um.

Tabelle 11.21: Stringfunktionen von MySQL (Forts.)

Die Folgenden Beispiele geben einen kurzen Einblick in die Arbeit mit den Stringfunktionen von MySQL:

```
SELECT UPPER(string) FROM test;
```

Ausgabe:

```
HALLO
WOLKENBRUCH
DAS IST EIN STRING
```

```
SELECT LEFT(string, 3) FROM test WHERE zahl = -2469.5;
```

Ausgabe:

```
Das
```

```
SELECT CONCAT(SPACE(30), REVERSE(string)) FROM test WHERE zahl = -
2469.5;
```

Ausgabe:

```
                              gnirtS nie tsi saD
```

```
SELECT LENGTH(string) FROM test;
```

Ausgabe:

```
5
11
18
```

```
SELECT INSERT(string, 5, 3, "war") FROM test WHERE zahl = -2469.5;
```

Ausgabe:

```
Das war ein String
```

Datums- und Zeitfunktionen

MySQL verfügt über unterschiedliche Datentypen für die Speicherung und Wiedergabe von Datums- und Zeitangaben. Speziell für diese Datentypen stellt die MySQL-Software eine Reihe von Funktionen zur Verfügung.

Für die folgenden Funktionen wird zwischen den Typen `DATE`, `YEAR`, `TIME` und `DATETIME` unterschieden.

Funktion	Beschreibung
`DAYOFWEEK(date)`	Gibt den Tag der Woche als Zahl zurück. Sonntag entspricht 1.
`DAYOFMONTH(date)`	Tag des Monats
`DAYOFYEAR(date)`	Tag im Jahr
`MONTH(date)`	Gibt den Monat zurück.
`YEAR(date)`	Gibt das Jahr zurück.
`DAYNAME(date)`	Gibt den Wochentag zurück (englisch).
`QUARTER(date)`	Gibt das Quartal zurück.
`WEEK(date)`	Gibt die Woche zurück.
`HOUR(date)`	Gibt die Stunde zurück.
`MINUTE(date)`	Gibt die Minute zurück.
`SECOND(date)`	Gibt die Sekunde zurück.
`TO_DAYS(date)`	Gibt die Zahl der Tage seit dem Jahr 0 zurück.
`FROM_DAYS(n)`	Ermittelt das Datum nach der Anzahl n der Tage.
`CURDATE()`	Gibt das aktuelle Datum zurück.
`CURTIME()`	Gibt die aktuelle Zeit zurück.
`NOW()`	Gibt das aktuelle Datum mit Zeit zurück.
`CURRENT_TIMESTAMP()`	Gibt das aktuelle Datum mit Zeit zurück.
`UNIX_TIMESTAMP()`	Gibt den aktuelle Timestamp zurück.
`UNIX_TIMESTAMP(date)`	Gibt den Timestamp für das Datum date zurück.
`SEC_TO_TIME(s)`	Gibt die Sekunden s im Format hh:mm:ss zurück.
`TIME_TO_SEC(time)`	Gibt die Zeitangabe in Sekunden zurück.

Tabelle 11.22: Datums- und Zeitfunktionen von MySQL

Im folgenden nun ein paar Beispiele für die Datums- und Zeitfunktionen von MySQL.

```
SELECT NOW();
```

Ausgabe:

```
2001-12-18 15:25:04
```

```
SELECT DAYNAME(datum) FROM test;
```

Ausgabe:

```
Monday
Thursday
Tuesday
```

```
SELECT TO_DAYS(datum) FROM test;
```

Ausgabe:

```
731208
677262
730453
```

```
SELECT MONTHNAME(datum) FROM test;
```

Ausgabe:

```
December
April
November
```

```
SELECT UNIX_TIMESTAMP(datum) FROM test;
```

Ausgabe:

```
1009148400
0
943916400
```

Warum hat das zweite Datum den Wert 0? Da der UNIX-Timestamp erst seit 1970 existiert, ist jedes Datum vor dieser Zeit nicht erfasst. MySQL gibt für nicht gültige Daten den Wert 0 aus.

Formatierte Ausgabe von Datum und Zeit

Neben den oben genannten Datums- und Zeitfunktionen unterstützt MySQL zwei weitere Funktionen, die es erlauben, die gewünschten Daten in jedem beliebigen Format zu erstellen. Für das Datum steht die Funktion `DATE_FORMAT()` zur Verfügung und für die Zeit die Funktion `TIME_FORMAT()`.

Beide Funktionen arbeiten mit denselben Argumenten:

```
DATE_FORMAT(date, format);
TIME_FORMAT(time, format);
```

Der erste Parameter wird mit einem der jeweiligen Datentypen gefüllt. Der zweite Parameter ist ein String, der sich aus folgenden Kürzeln zusammensetzt.

Formatzeichen	Erklärung
%M	Monatsname (englisch)
%m	Monat nummerisch mit führender 0
%b	Abgekürzter Monatsname (englisch)
%c	Monat nummerisch ohne führende 0
%W	Wochentag (englisch)
%w	Tag der Woche (0=Sonntag, 1= Montag)
%a	Abgekürzter Wochennamen (englisch)
%D	Tag des Monats mit engl. Suffix
%d	Tag des Monats mit führender 0
%e	Tag des Monats ohne führende 0
%Y	Jahr vierstellig
%y	Jahr zweistellig
%j	Tage des Jahres
%H	Stunden mit führender 0
%k	Stunden ohne führende 0
%h	Stunden von 1 bis 12 mit führender 0
%l	Stunden von 1 bis 12
%i	Minuten
%r	Komplette Zeitangabe von 1 bis 12
%T	Komplette Zeitangabe von 0 bis 23
%S	Sekunden mit führender 0

Tabelle 11.23: Formatzeichen für DATE_FORMAT und TIME_FORMAT

Formatzeichen	Erklärung
%s	Sekunden ohne führende 0
%p	AM oder PM
%U	Woche im Jahr, Sonntag ist der erste Tag einer Woche.
%u	Woche im Jahr, Montag ist der erste Tag einer Woche.

Tabelle 11.23: Formatzeichen für DATE_FORMAT und TIME_FORMAT (Forts.)

Der Formatstring kann alle Buchstaben und Zeichen enthalten, die MySQL untersützt. Alle Angaben werden genauso wiedergegeben, wie sie im Formatstring angegeben wurden. Lediglich die Formatzeichen werden mithilfe der Funktion durch den jeweiligen Wert ersetzt.

```
SELECT DATE_FORMAT("2001-4-24", "%W, the %D. %M %Y");
```

Ausgabe:

```
Tuesday, the 24th. April 2001
```

```
SELECT TIME_FORMAT("16:18:43", "Es ist %h %i %s %p");
```

Ausgabe:

```
Es ist 04 18 43 PM
```

Die Funktionen DATE_ADD() und DATE_SUB()

Mit den Funktionen `DATE_ADD()` und `DATE_SUB()` können Datumsberechnungen durchgeführt werden. MySQL erlaubt es, auf diesem Wege in einem virtuellen Kalender in bestimmten Einheiten vor- oder zurückzugehen. Neben einem gültigen Datum erwarten beide Funktionen einen weiteren Parameter:

```
DATE_ADD(date, INTERVAL ausdruck typ);
DATE_SUB(date, INTERVAL ausdruck typ);
```

Die Parameter »ausdruck« und »typ«, die nach dem Schlüsselwort INTERVAL angegeben werden müssen, geben das Zeitintervall an, mit dem die Funktion arbeiten soll. `DATE_ADD()` rechnet das angegebene Intervall zum übergebenen Datum dazu. Im Gegensatz dazu zieht die Funktion `DATE_SUB()` das angegebene Intervall vom Datum ab.

Die möglichen Angaben für die Intervall-Parameter können Sie aus dieser Tabelle entnehmen:

Format (ausdruck)	Typ (typ)	Beschreibung	Beispiel
ss	SECOND	Sekunden	04 SECOND
mm	MINUTE	Minuten	33 MINUTE
hh	HOUR	Stunden	01 HOUR
DD	DAY	Tage	12 DAY
MM	MONTH	Monate	05 MONTH
YY	YEAR	Jahre	99 YEAR
»mm:ss«	MINUTE_SECOND	Minuten und Sekunden	»42:04« MINUTE_SECOND
»hh:mm«	HOUR_MINUTE	Stunden und Minuten	»03:12« HOUR_MINUTE
»DD hh«	DAY_HOUR	Tage und Stunden	»24 18« DAY_HOUR
»YY-MM«	YEAR_MONTH	Jahre und Monate	»45-03« YEAR_MONTH
»hh:mm:ss«	HOUR_SECOND	Stunden, Minuten, Sekunden	»12:34:56« HOUR_SECOND
»DD hh:mm«	DAY_MINUTE	Tage, Stunden, Minuten	»23 12:36« DAY_MINUTE
»DD hh:mm:ss«	DAY_SECOND	Tage, Stunden, Minuten, Sekunden	»21 21:45:02« DAY_SECOND

Tabelle 11.24: Parameter für die DATE_ADD() und DATE_SUB()

Das folgende Beispiel zeigt, wie zu bestehenden Datumsangaben 91 Tage durch die Funktion `DATE_ADD()` dazu addiert werden.

```
SELECT DATE_ADD(datum, INTERVAL 91 DAY) FROM test;
```

Ausgabe:

```
2002-03-25
1854-07-13
2000-02-29
```

Das zweite Beispiel macht Gebrauch von der Funktion `DATE_SUB()` und zieht drei Jahre und vier Monate von jedem Datensatz ab.

```
SELECT DATE_SUB(datum, INTERVAL "03-04" YEAR_MONTH) FROM test;
```

Ausgabe:

```
1998-08-24
1850-12-13
1996-07-30
```

Aggregatfunktionen

Aggregatfunktionen dienen dazu, komplette Spalten einer Tabelle auszuwerten und einen Funktionswert daraus zu berechnen. Im Gegensatz zu den Funktionen, die immer nur einen Wert als Argument übernehmen können, kann eine Aggregatfunktion alle Werte einer Anfrage auf einmal mit in die Berechnung einbeziehen.

MySQL stellt folgende Aggregatfunktionen zur Verfügung:

Funktion	Beschreibung
`COUNT(*)`	Gibt die Anzahl aller Spalten einer Ergebnisrelation zurück
`COUNT(name)`	Gibt die Anzahl der Felder der Spalte name zurück, die in einer Ergebnisrelation zusammengefasst wurden. Felder mit dem Wert NULL werden nicht berücksichtigt.
`AVG(name)`	Berechnet den Mittelwert einer Spalte
`MIN(name)`	Gibt den kleinsten Wert einer Spalte zurück
`MAX(name)`	Gibt den größten Wert einer Spalte zurück
`SUM(name)`	Gibt die Summe aller Felder einer Spalte zurück
`STD(name)`	Gibt die Standardabweichung einer Spalte zurück

Tabelle 11.25: Aggregatfunktionen

Die folgenden Beispiele demonstrieren die wichtigsten Aggregatfunktionen am Beispiel der Tabelle »test«.

```
SELECT COUNT(*) FROM test;
```

Ausgabe:

```
3
```

```
SELECT AVG(zahl) FROM test;
```

Ausgabe:

```
2600.22500102
```

```
SELECT MAX(zahl) FROM test;
```

Ausgabe:

```
10143
```

```
SELECT SUM(zahl) FROM test;
```

Ausgabe:

```
7800.6750
```

11.3.7 Übungen

Ü

1. Erklären Sie den Unterschied zwischen dem externen und dem internen Schema einer Datenbank.
2. Was ist eine Relation?
3. Was ist der Unterschied zwischen SQL und MySQL?
4. Wofür steht der Ausdruck NULL?
5. Was ist der Unterschied zwischen CHAR und VARCHAR?
6. Was ist ein Schlüssel?
7. Was ist der Unterschied zwischen UPDATE und REPLACE?
8. Was ist falsch an dieser Anweisung?

   ```
   SELECT * FROM telnum WHERE name == "Dirk";
   ```

9. Können mathematische Funktionen auch in anderen nicht-MySQL-Datenbanken angewendet werden?
10. Was unterscheidet Aggregatfunktionen von anderen Funktionen?

11.4 PHP und die MySQL-Datenbank

In den letzten Kapiteln haben Sie die wichtigsten Bestandteile der Sprache SQL kennen gelernt, die zu den wesentlichen Schlüsseln einer relationalen Datenbank gehört. Bisher wurden alle Anfragen an die Zieldatenbank direkt oder über eine bestehende Schnittstelle gestellt, die es erlaubt, auf den Datenbestand der Datenbank zuzugreifen. Der Nachteil dieser Technik ist die Abhängigkeit von bestimmten Tools, die eine Schnittstelle zwi-

schen User und den Datensätzen schaffen. Sie sind gezwungen alle Anfragen nach den Anforderungen dieser Software zu richten und haben keine Möglichkeit dynamisch zu arbeiten.

In den folgenden Kapiteln werden Sie lernen, wie Sie mit PHP eine eigene Schnittstelle zu einer MySQL-Datenbank öffnen und Webseiten dynamisch mit den gespeicherten Information erschaffen. Das nötige PHP-Modul ist wegen der immensen Wichtigkeit der MySQL Datenbank von Anfang an in PHP4 integriert und muss nicht nachträglich installiert werden.

Weiterführende Informationen zu MySQL finden Sie unter *http://www.mysql.com*.

11.4.1 Verbindung zur Datenbank erstellen

Der erste Schritt im Umgang mit einer Datenbank ist eine erfolgreiche Connection zum MySQL-Datenbankserver. Dazu benötigen Sie die Funktion `mysql_connect()`, die eine Verbindung zum Datenbankserver erstellt und einen User anmeldet.

```
$dbh = mysql_connect($server, $user, $pass);
```

Die Funktion erwartet drei Parameter, die für die Verbindung notwendig sind. Der erste Parameter ist der Name des Servers, auf dem die MySQL-Datenbank installiert ist. Alternativ zu einem Hostnamen kann auch eine gültige IP-Adresse angegeben werden. Liegt die Datenbank auf demselben Server wie der PHP-Interpreter, kann für den Server auch »localhost« angegeben werden.

Der zweite und dritte Parameter sind die Userdaten des MySQL-Users. Sie bekommen diese Daten von Ihrem Administrator zugewiesen. Über diese Zugangsdaten wird in der MySQL-Software festgelegt, welche Rechte Sie in der Datenbank besitzen. Falls Sie mehr über die Zugriffsrechte innerhalb von MySQL erfahren möchten, dann lesen Sie bitte im Kapitel über »Das Arbeitsumfeld« von PHP nach.

War die Connection zum MySQL-Server erfolgreich, dann gibt die Funktion `mysql_connect()` einen Verbindungshandle auf die Datenbank zurück. Diese Information muss bei allen weiteren Funktionen angegeben werden. Wenn der Vorgang nicht erfolgreich war, gibt die Funktion FALSE zurück.

```
<?
$server = "localhost";
$user = "root";
$pass = "";

$dbh = mysql_connect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!";}
?>
```

Listing 11.4: Verbindung zum MySQL-Server

Ausgabe:

```
mySQL-Verbindung erfolgreich!
```

Das Beispielskript stellt die Verbindung zu einer Datenbank auf dem Server »localhost« her. Die Userdaten und der Hostname werden in Variablen an die `mysql_connect()`-Funktion übergeben.

Das Gegenstück zu `mysql_connect()` ist die Funktion `mysql_close()`, die die Verbindung zu einem MySQL-Server wieder schließt. Als einzigen Parameter erwartet die Funktion einen gültigen Handle auf eine MySQL-Verbindung.

```
mysql_close($dbh);
```

Die Verbindung zu einem Datenbankserver wird am Ende eines Skripts automatisch geschlossen. Im Zweifelsfall ist es also nicht nötig, die Funktion explizit aufzurufen.

Die Funktion mysql_pconnect()

Alternativ zur Funktion `mysql_connect()` kann auch die Funktion `mysql_pconnect()` verwendet werden, die versucht eine persistente Verbindung zu einem MySQL-Server zu erstellen. Voraussetzung für diese Funktion ist allerdings, dass der PHP-Interpreter als Apache-Modul im Webserver integriert ist und nicht als CGI.

Eine persistente Verbindung hat den Vorteil, dass beim Aufruf einer neuen PHP-Seite nicht erneut eine MySQL-Verbindung aufgebaut werden muss. Eine persistente Verbindung bleibt über die Laufzeit des Skripts bestehen und kann jederzeit wieder von einer neuen Anfrage genutzt werden. Die erneute Nutzung einer Verbindung erfolgt über die Angabe der identischen Parameter bei der Verbindung mit `mysql_pconnect()`. Die Funktion `mysql_close()` hat keinen Einfluss auf eine Verbindung, die mit `mysql_pconnect()` erstellt wurde.

```
<?
$server = "localhost";
$user = "root";
$pass = "";

$dbh = mysql_pconnect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!";}
?>
```

Listing 11.5: MySQL-Verbindung mit mysql_pconnect()

Ausgabe:

```
mySQL-Verbindung erfolgreich!
```

Die Funktion arbeitet mit denselben Parametern wie die Funktion `mysql_connect()` und kann auch im selben Context verwendet werden. Das zurückgegebene Handle ist für alle MySQL-Funktionen gültig.

11.4.2 Verwaltungsfunktionen

Bevor wir damit beginnen, die Datenbankverbindung für SQL-Befehle zu nutzen, möchte ich Ihnen einige Verwaltungsfunktionen vorstellen, die für die Arbeit mit einer MySQL-Datenbank recht nützlich sind.

Funktion	Beschreibung
`mysql_change_user()`	Der aktuelle User wird abgemeldet und ein neuer User wird angemeldet.
`mysql_select_db()`	Die aktuelle Datenbank wird ausgewählt bzw. gewechselt.
`mysql_create_db()`	Eine neue Datenbank wird erstellt.
`mysql_drop_db()`	Eine Datenbank wird gelöscht.

Tabelle 11.26: Verwaltungsfunktionen

Die Funktion `mysql_change_user()` wird dazu verwendet den User innerhalb eines Skripts zu wechseln, falls das nötig sein sollte. Zu diesem Zweck müssen vier Parameter übergeben werden: zuerst der Username des neuen Users inklusive dem dazugehörigen Passwort. Als weiterer Parameter muss ein Datenbankname angegeben werden sowie der aktuelle Handle auf die Datenbankconnection.

```
$bool = mysql_change_user($user, $passwd, $dbase, $dbh);
```

Die Funktion meldet den Benutzer der aktuellen Verbindung bei der Datenbank neu an und setzt die aktive Datenbank auf $dbase. Der Wechsel des Users mit dieser Funktion ist erst ab der MySQL-Version 3.23.3 möglich. Wenn der Versuch misslingt, gibt die Funktion FALSE zurück und die aktuelle Verbindung bleibt bestehen.

Mit der Funktion `mysql_select_db()` ist es möglich, die aktuelle Datenbank festzulegen, bzw. zu wechseln. Die Funktion erwartet lediglich zwei Parameter, nämlich den Namen der Zieldatenbank sowie das aktuelle Verbindungshandle.

```
$bool = mysql_select_db($dbname, $dbh);
```

Wenn die Funktion erfolgreich ausgeführt wurde, gibt die Funktion TRUE zurück, ansonsten den Wert FALSE. Die Funktion entspricht der Anweisung USE in SQL, die intern auch in der Datenbank ausgeführt wird.

Datenbanken erstellen und löschen

Neben den bereits vorgestellten Funktionen verfügt das MySQL-Modul noch über zwei weitere Funktionen, mit denen es möglich ist, direkt aus PHP Datenbanken zu erstellen und wieder zu löschen. Sowohl die Funktion `mysql_create_db()` als auch die Funktion `mysql_drop_db()` erwarten zwei Parameter: einmal den Namen der Zieldatenbank und als zweiten Punkt einen Handle auf eine Datenbankverbindung.

```
$bool = mysql_create_db($name, $dbh);
$bool = mysql_drop_db($name, $dbh);
```

Die Funktion `mysql_create_db()` entspricht der SQL-Anweisung `CREATE DATABASE`. Sie erstellt eine neue Datenbank auf dem ausgewählten MySQL-Server. Dem steht die Funktion `mysql_drop_db()` gegenüber. Sie entspricht der SQL-Anweisung `DROP DATABASE` und löscht eine bestehende Datenbank mit allen darin gespeicherten Daten.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "temp";

$dbh = mysql_pconnect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!<br>";}

$bool = mysql_create_db($db, $dbh);
if ($bool) {echo "Datenbank $db wurde erstellt!<br>";}
```

```
$bool = mysql_drop_db($db, $dbh);
if ($bool) {echo "Datenbank $db wurde gelöscht!";}
?>
```

Listing 11.6: Die Funktion mysql_create_db() und mysql_drop_db()

Ausgabe:

```
mySQL-Verbindung erfolgreich!
Datenbank temp wurde erstellt!
Datenbank temp wurde gelöscht!
```

Sowohl die Funktion `mysql_create_db()` als auch die Funktion `mysql_drop_db()` geben bei Erfolg TRUE zurück. Sollte die Datenbank nicht erstellt oder gelöscht werden können, wird dieser Fehler mit FALSE quittiert.

11.4.3 Die Datenbankverbindung nutzen

Nachdem die Verbindung zum MySQL-Datenbankserver erstellt wurde, steht über das Verbindungshandle der Zugriff auf alle vorhandenen Datenbanken offen. PHP unterstützt zwei verschiedene Funktionen, die es ermöglichen, SQL-Anweisungen an die geöffneten Datenbanken zu schicken.

Funktion	Beschreibung
`mysql_query()`	Die Funktion sendet eine SQL-Anfrage an die aktuelle Datenbank.
`mysql_db_query()`	Die Funktion sendet eine SQL-Anfrage an eine Datenbank, die als Parameter übergeben wird.

Tabelle 11.27: MySQL-Zugriffsfunktion

Da die MySQL-Datenbank von außen nur über eine SQL-Schnittstelle erreicht werden kann, sind diese beiden Funktionen die einzigen Möglichkeiten generellen Zugriff auf die Datensätze zu erhalten. In einem späteren Kapitel werden Sie noch weitere Funktionen kennen lernen, die für spezielle Zugriffe entwickelt worden sind.

Sowohl `mysql_query()` als auch `mysql_db_query()` kommunizieren über SQL mit der geöffneten Datenbank. Jeder Zugriff muss also zuvor SQL-codiert werden und als String der jeweiligen Funktion übergeben werden. `mysql_query()` ist die einfachere der beiden Funktionen, da sie keinen Parameter bezüglich der Zieldatenbank erwartet.

```
$erg = mysql_query($query, $dbh);
```

Die Funktion übernimmt als ersten Parameter einen String, der die SQL-Anweisung für die Datenbankanfrage enthält. Der zweite Parameter übergibt das Datenbankhandle, das von der Funktion `mysql_connect()` bzw. `mysql_pconnect()` erstellt wurde.

Anweisungen, die auf `INSERT`, `UPDATE` oder `DELETE` beruhen, geben TRUE zurück, wenn der Zugriff erfolgreich war. SELECT-Anweisungen geben einen Zeiger auf die Ergebnisrelation zurück, die im Speicher abgelegt wurde.

```
<?
$server = "localhost";
$user = "root";
$pass = "";

$dbh = mysql_pconnect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!<br>";}

$query = "SELECT * FROM name.telnum";

$erg = mysql_query($query, $dbh);
if ($erg) {echo "SQL-Anweisung erfolgreich ausgeführt!";}
?>
```

Listing 11.7: Die Anweisung mysql_query()

Ausgabe:

```
mySQL-Verbindung erfolgreich!
SQL-Anweisung erfolgreich ausgeführt!
```

Wenn keine Datenbank ausgewählt wurde oder auf eine andere Datenbank zugegriffen werden soll, kann dies ganz einfach über die SQL-Notation `datenbank.tabelle` realisiert werden.

Die Funktion `mysql_db_query()` arbeitet nach demselben Muster wie die Funktion `mysql_query()`. Der Unterschied liegt darin, dass `mysql_db_query()` die Zieldatenbank explizit festlegt und deshalb einen weiteren Parameter fordert.

```
$erg = mysql_db_query($db, $query, $dbh);
```

Neben der SQL-Anweisung und dem Connection-Handle muss ein String mit dem Namen der Zieldatenbank übergeben werden. Genau wie `mysql_query()` gibt die Funktion entweder TRUE oder einen Zeiger auf die Ergebnisrelation im Speicher zurück.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!<br>";}

$query = "SELECT * FROM name.telnum";

$erg = mysql_db_query($db, $query, $dbh);

if ($erg) {echo "SQL-Anweisung erfolgreich ausgeführt!";}
?>
```

Listing 11.8: Die Funktion mysql_db_query()

Unter Umständen gibt der PHP-Interpreter (je nach Version) eine Warnung an dieser Stelle aus, dass die Funktion `mysql_db_query()` veraltet (deprecated) sei. Alternativ solle man die Funktion `mysql_query()` in Verbindung mit der Funktion `mysql_select_db()` verwenden, um denselben Erfolg zu erzielen. Im Zweifelsfall ist es also besser, die Funktion `mysql_query()` zu verwenden, da sie up-to-date ist.

Egal welche Funktion Sie verwenden: Sobald die SQL-Anweisung eine SELECT-Anfrage ist, wird das Ergebnis als eine Tabelle im Speicher abgelegt. Die Rückgabe der jeweiligen Funktion ist ein Zeiger auf das Ergebnis im Speicher, der einer Auswertungsfunktion übergeben werden muss.

Auswertung von Ergebnisrelationen im Speicher

Die Problematik von Ergebnisrelationen einer SQL-Anfrage besteht in der Struktur der zurückgegebenen Daten. Im Gegensatz zu den bekannten Funktionen werden hier nicht einzelne Werte oder Arrays zurückgegeben, sondern komplexe Datenstrukturen in der Form von zweidimensionalen Feldern.

Es gibt verschiedene Möglichkeiten an dieses Problem heranzugehen: Das MySQL-Modul von PHP hat einige in seiner Funktionalität berücksichtigt und stellt verschiedene Funktionen zur Verfügung, die in der Lage sind das Ergebnis im Speicher auszuwerten.

Funktion	Beschreibung
`mysql_fetch_row()`	Holt einen Datensatz aus der Ergebnisliste im Speicher als nummerisches Array.
`mysql_fetch_array()`	Holt einen Datensatz aus der Ergebnisliste im Speicher als assoziatives Array und nummerisches Array.
`mysql_fetch_assoc()`	Holt einen Datensatz aus der Ergebnisliste im Speicher nur als assoziatives Array.
`mysql_result()`	Gibt den Inhalt eines Feldes aus der Ergebnisliste zurück.
`mysql_affected_rows()`	Gibt die Anzahl der von der letzten Operation betroffenen Reihen zurück.
`mysql_free_result()`	Gibt den Speicher der Ergebnisliste frei.

Tabelle 11.28: Funktionen zur Auswertung von Ergebnislisten

Die drei wichtigsten Funktionen im Umgang mit Ergebnislisten im Speicher sind die Funktionen `mysql_fetch_row()`, `mysql_fetch_array()` und `mysql_fetch_assoc()`. Alle drei arbeiten nach demselben Prinzip: jeder Aufruf einer dieser Funktionen gibt eine Reihe (nicht Spalte!) der Ergebnisliste zurück. Der erste Aufruf gibt die erste Reihe zurück, der zweite Aufruf die zweite Reihe und so weiter ...

PHP merkt sich automatisch, welche Reihe als Letztes ausgelesen wurde, indem ein interner Zeiger auf die jeweils aktuelle Reihe gesetzt wird. Jede der drei Funktionen gibt ein eindimensionales Array mit den Werten der jeweiligen Ergebnisreihe zurück. Wurde über das Ende der Tabelle hinausgelesen, dann gibt jede Funktion FALSE zurück.

Die drei Funktionen sind vom Aufbau her gleich gegliedert und unterscheiden sich nur im Detail. Welche Funktion letztendlich eingesetzt wird, bleibt Geschmackssache und ist jedem selbst überlassen.

Die Funktion `mysql_fetch_row()` ist die einfachste der drei Möglichkeiten. Sie erwartet als einzigen Parameter einen Zeiger auf die Ergebnisrelation im Speicher und gibt ein nummerisches Array der aktuellen Tabellenzeile zurück.

```
$liste = mysql_fetch_row($erg);
echo $liste[1];
```

Ausgabe:

```
Dirk
```

Dem gegenüber steht die Funktion `mysql_fetch_assoc()`, die ebenfalls nur einen Parameter erwartet. Der Unterschied zu `mysql_fetch_row()` besteht darin, dass diese Funktion ein assoziatives Array zurückgibt. Die Bezeichner der jeweiligen Elemente werden aus den Spaltennamen der Ergebnisrelation abgeleitet.

```
$liste = mysql_fetch_assoc($erg);
echo $liste["name"];
```

Ausgabe:

```
Dirk
```

Die flexibelste der drei Funktionen ist `mysql_fetch_array()`, die einem die Wahl lässt, ob das Ergebnisarray nummerisch oder assoziativ erscheinen soll. Über einen optionalen Parameter ist es möglich, beim Funktionsaufruf festzulegen, welche der beiden Lösungen genutzt werden soll. Wird der Parameter weggelassen, dann stehen beide Arrays zur Verfügung.

```
$liste = mysql_fetch_array($erg, MYSQL_BOTH);
echo $liste["name"];
echo "<br>";
echo $liste[2];
```

Ausgabe:

```
Dirk
1234
```

Der optionale Parameter nach dem Zeiger auf die Ergebnisliste kann folgende Werte annehmen:

- `MYSQL_ASSOC`: Erzeugt ein assoziatives Array.
- `MYSQL_NUM`: Erzeugt ein nummerisches Array.
- `MYSQL_BOTH`: Erzeugt beides; Standardwert.

Da alle drei Funktionen immer nur eine Zeile der Ergebnistabelle auslesen, bietet es sich an, durch eine Schleife die komplette Relation auf einmal auszulesen. Wenn nicht gerade nach einem bestimmten Einzelwert im Ergebnis gesucht wird, dann ist dies die gängigste Methode in der Praxis.

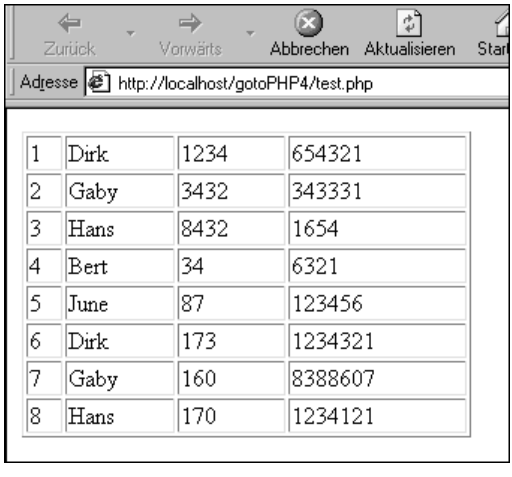
```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT * FROM telnum";
$erg = mysql_query($query, $dbh);

echo "<table border=1 width=300>";
while ($l = mysql_fetch_assoc($erg))
{
   echo "<tr>";
   foreach ($l as $var)
   {
      echo "<td>$var</td>";
   }
   echo "</tr>";
}
echo "</table>";
?>
```

Listing 11.9: Auswertung einer MySQL-Relation

Ausgabe:



1	Dirk	1234	654321
2	Gaby	3432	343331
3	Hans	8432	1654
4	Bert	34	6321
5	June	87	123456
6	Dirk	173	1234321
7	Gaby	160	8388607
8	Hans	170	1234121

Abbildung 11.2: Die Ergebnisrelation im Speicher

Die einfachste Möglichkeit, die komplette Ergebnisliste auszulesen, ist eine `while()`-Schleife, die so lange läuft, bis die Rückgabe von `mysql_fetch_assoc()` FALSE ist. Das gilt natürlich auch für die Funktionen `mysql_fetch_row()` und `mysql_fethc_array()`. Der Rest des Programms ist trivial: Eine `foreach()`-Schleife wertet jedes Array aus und gibt so zeilenweise die Daten der Ergebnisrelation an den Browser aus. Über eine einfache HTML-Tabelle wird das Ergebnis strukturiert.

In der Praxis ist es oft der Fall, dass die Daten einer SQL-Anfrage nicht sofort im Browser ausgegeben werden sollen, sondern erst zwischengespeichert werden. Die naheliegendste Möglichkeit die Daten übersichtlich und gleichzeitig strukturtreu zu speichern, ist ein zweidimensionales Array, das den kompletten Aufbau der Ergebnisrelation nachahmt.

Die Erstellung eines solches Arrays ist relativ leicht zu bewerkstelligen und kann die Grundlage des letzten Beispielskripts nutzen:

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT * FROM telnum";
$erg = mysql_query($query, $dbh);

$i=0;
while ($l = mysql_fetch_assoc($erg))
{
   $data[$i] = $l;
   $i++;
}
echo $data[0]["name"];
echo "<br>";
echo $data[2]["telefon"];
echo "<br>";
echo $data[5]["id"];
?>
```

Listing 11.10: Das Ergebnis wird in einem assoziativen Array abgelegt

Ausgabe:

```
Dirk
1654
6
```

Wie das Beispiel zeigt, können die Daten relativ leicht gehandhabt werden, indem man sich an der Struktur der Ergebnisrelation orientiert. Das assoziative Array wird einfach über eine weitere Variable erstellt, die bei jedem Durchgang inkrementiert wird. Dieser Wert kann als Offset des Arrays genutzt werden, so dass jede Zeile als neuer Wert übergeben werden kann.

Neben den allgemeinen Funktionen zur Auswertung einer Ergebnisrelation stellt das MySQL-Modul von PHP noch eine weitere Funktion zur Verfügung, um einzelne Werte aus dem Speicher auszulesen. Die Funktion `mysql_result()` gibt den Inhalt genau eines Feldes zurück.

```
$data = mysql_result($erg, $row, $col);
```

Die Funktion erwartet drei Parameter: einmal den Zeiger auf die Ergebnisrelation im Speicher sowie zwei Angaben zum Zielwert in der Tabelle. Die Werte `$row` und `$col` werden als Integerwerte angegeben und beziehen sich damit als eine Art Koordinaten auf die Reihe und die Spalte der Ergebnistabelle. Alternativ kann der Wert `$col` auch als String übergeben werden, indem er den Namen der Ergebnisspalte angibt.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT * FROM telnum";
$erg = mysql_query($query, $dbh);

$row = 0;
$col = "name";
$data = mysql_result($erg, $row, $col);
echo $data;
?>
```

Ausgabe:

```
Dirk
```

Die jeweils erste Reihe und erste Spalte werden mit der Nummer 0 angesprochen. Im Gegensatz zu den Funktionen `mysql_fetch_row()`, `mysql_fetch_assoc()` und `mysql_fetch_array()` bietet die Funktion `mysql_result()` bei weitem nicht den Komfort und die Flexibilität, um große Mengen von Datensätzen zu analysieren. Sie sollte deshalb nur eingesetzt werden, wenn es darum geht, einzelne Daten aus einer Relation auszulesen.

Zu guter Letzt möchte ich eine Funktion vorstellen, die für die Aufräumarbeiten nach der Auswertung der Ergebnisrelation im Speicher zuständig ist. Damit keine unnötigen Ressourcen auf dem Rechner belegt werden, stellt PHP die Funktion `mysql_free_result()` zur Verfügung, die es ermöglicht, den Speicherplatz der Ergebnisrelation wieder freizugegeben.

```
$bool = mysql_free_result($erg);
```

Als einzigen Parameter erwartet `mysql_free_result()` den Zeiger auf das Ergebnis, das im Speicher gelöscht werden soll. War der Vorgang erfolgreich, gibt die Funktion TRUE zurück, ansonsten FALSE.

Da der Speicher am Ende eines Skripts immer automatisch freigegeben wird, ist die Anwendung also nur innerhalb eines Skripts sinnvoll.

Die Funktion mysql_affected_rows()

Die Funktion `mysql_affected_rows()` gibt die Anzahl der von der letzten Operation betroffenen Reihen zurück. So ist es möglich nachzuvollziehen, welche Auswirkung eine Anweisung auf die Zieltabelle gehabt hat.

```
$num = mysql_affected_rows($dbh);
```

Die Funktion erwartet als Parameter das Verbindungshandle zur geöffneten Datenbank. Ein häufiger Fehler ist hier die Annahme, dass die Funktion den Zeiger auf die Ergebnisrelation im Speicher erwartet, um die Anzahl der betroffenen Reihen bestimmen zu können. Das ist nicht der Fall: Es wird lediglich das aktuelle Datenbankhandle übergeben.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);
```

```
$query = "SELECT * FROM telnum";
$erg = mysql_query($query, $dbh);

$num = mysql_affected_rows($dbh);
echo "$num Reihen sind von der Funktion betroffen!";
?>
```

Listing 11.11: Die Funktion mysql_affected_rows()

Ausgabe:

```
8 Reihen sind von der Funktion betroffen!
```

Wurde die zuvor beabsichtigte Anweisung nicht korrekt ausgeführt (Syntaxfehler, etc.), gibt die Funktion `-1` zurück (nicht FALSE!).

Informationen über die Ergebnisliste

Ähnlich wie die Funktion `mysql_affected_rows()` Zahlen über die betroffenen Datensätze in der Datenbank ermittelt, gibt es auch Funktionen, die es ermöglichen, Informationen über die erschaffene Ergebnisrelation im Speicher auszulesen. Dabei handelt es sich um Werte, die Auskunft darüber geben, welchen Umfang die Tabelle im Speicher hat und wie viele Datensätze in ihr gespeichert sind.

Die Funktion `mysql_num_rows()` gibt die Anzahl der Datensätze, also der Reihen, der Ergebnisrelation im Speicher zurück. Als einzigen Parameter muss der Zeiger auf die Tabelle übergeben werden.

```
$num = mysql_num_rows($erg);
```

Das Ergebnis wird in Form eines Integerwertes zurückgegeben. Wenn die Ergebnisliste keine Datensätze enthält, wird der Wert 0 ausgegeben.

Die zweite Funktion, die Auskunft über die Größe der Ergebnisrelation gibt, ist die Funktion `mysql_num_fields()`. Im Gegensatz zur Funktion `mysql_num_rows()` ermittelt sie die Anzahl der Spalten der Tabelle im Speicher.

```
$fields = mysql_num_fields($erg);
```

Auch diese Funktion gibt den Wert 0 zurück, falls die Ergebnisrelation keine Datensätze enthält. In der Praxis sieht die Anwendung der beiden Funktionen so aus:

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";
$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT * FROM test";
$erg = mysql_query($query, $dbh);

$rows = mysql_num_rows($erg);
$fields = mysql_num_fields($erg);
echo "Die Ergebnisrelation hat $rows Datensätze und $fields Spal-
ten!";
?>
```

Listing 11.12: Die Funktionen mysql_num_rows() mysql_num_fields()

Ausgabe:

```
Die Ergebnisrelation hat 3 Datensätze und 3 Spalten!
```

In der Ergebnisliste navigieren

Immer wenn eine PHP-Funktion auf die Ergebnisrelation im Speicher zugreift, wird der interne Zeiger um eine Zeile weiter versetzt. Auf diesem Wege merkt sich der PHP-Interpreter, auf welche Daten zuletzt im Speicher zugegriffen wurde. Auf diese Weise ist es nicht möglich, denselben Datensatz mehrmals auszulesen.

In manchen Fällen ist es allerdings nötig, genau das zu tun: Aufgrund dessen stellt das MySQL-Modul die Funktion `mysql_data_seek()` zur Verfügung, die es erlaubt, den Zeiger innerhalb der Ergebnisrelation zu bewegen. Das ganze System ist durchaus vergleichbar mit der Arbeit des Dateizeigers, der innerhalb einer Datei das aktuelle Zeichen markiert.

```
$bool = mysql_data_seek($erg, 0);
```

Der erste Parameter übergibt den Zeiger auf die Zielrelation im Speicher. Der zweite Parameter ist ein Integerwert, der die neue Zeile im Speicher festlegt. Wird der Wert 0 übergeben, dann wird der Zeiger auf die erste Zeile der Ergebnisrelation gesetzt. Wie gehabt gibt die Funktion bei Erfolg TRUE und ansonsten FALSE zurück.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";

$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT zahl FROM test ORDER BY zahl DESC";
$erg = mysql_query($query, $dbh);

//größte Zahl wird festgestellt
$max = mysql_result($erg, 0, 0);
echo "Die größte Zahl ist: $max<br><br>";

//Zeiger wird wieder auf den ersten Wert gesetzt
mysql_data_seek($erg, 0);

echo "Alle Zahlen:<br>";
while ($l = mysql_fetch_assoc($erg))
{
   echo $l["zahl"];
   echo "<br>";
}
?>
```

Listing 11.13: Die Funktion mysql_data_seek()

Ausgabe:

```
Die größte Zahl ist: 10143.0000

Alle Zahlen:
10143.0000
127.1750
-2469.5000
```

Das Skript wertet die bekannte Tabelle »test« in der Datenbank »name« aus. Zuerst wird der größte Wert bestimmt und ausgegeben, dann werden in einem zweiten Schritt alle Werte der Tabelle aufgezählt. Ohne die Funktion `mysql_data_seek()` würde die erste Zahl in der Liste fehlen.

Informationen über Datenbanken und Tabellen

Sobald die Verbindung zu einer MySQL-Datenbank erstellt wurde, können über den Datenbankhandle eine Reihe von Informationen über die

bestehenden Datenbanken und Tabellen erfragt werden. PHP stellt zu diesem Zweck einige Funktionen zur Verfügung:

Funktion	Beschreibung
`mysql_list_fields()`	Gibt Informationen über eine Tabelle zurück.
`mysql_list_dbs()`	Gibt eine Liste mit allen Datenbanken zurück.
`mysql_list_tables()`	Gibt Informationen über alle Tabellen einer Datenbank zurück.
`mysql_tablename()`	Wertet das Ergebnis der Funktion `mysql_list_tables()` aus.
`mysql_db_name()`	Wertet das Ergebnis der Funktion `mysql_list_dbs()` aus.
`mysql_field_name()`	Wertet das Ergebnis der Funktion `mysql_list_fields()` aus und ermittelt den Namen des Feldes.
`mysql_field_table()`	Wertet das Ergebnis der Funktion `mysql_list_fields()` aus und ermittelt den Namen der Tabelle.
`mysql_field_type()`	Wertet das Ergebnis der Funktion `mysql_list_fields()` aus und ermittelt den Typ des Feldes in der Tabelle.
`mysql_field_flags()`	Wertet das Ergebnis der Funktion `mysql_list_fields()` aus und ermittelt die Optionen des Feldes.

Tabelle 11.29: Informationsfunktionen

Die Funktionen `mysql_list_fields()`, `mysql_list_dbs()` und `mysql_list_tables()` arbeiten alle nach dem gleichen Schema: Sie werten die Daten der Zieldatenbank aus und geben das Ergebnis als Ergebnisrelation zurück. Genau wie bei regulären SQL-Anfragen wird über die Funktion ein Zeiger auf die Tabelle im Speicher zurückgegeben. Je nach eingesetzter Funktion wird durch eine spezielle Auswertungsfunktion das Ergebnis aus dem Speicher ausgelesen und im Programm verfügbar gemacht.

Jede der drei Funktionen arbeitet auf einer bestimmten Ebene und gibt die entsprechenden Informationen zurück. Die oberste Ebene ist die globale Übersicht über alle existierenden Datenbanken, die mit der Funktion `mysql_list_dbs()` ermöglicht wird.

```
$erg = mysql_list_dbs($dbh);
```

Als einzigen Parameter erwartet die Funktion das Verbindungshandle auf die geöffnete Datenbank. Der zurückgegebene Zeiger sollte in einer Vari-

able gespeichert werden. Die Auswertung der Daten erfolgt über die Funktion `mysql_db_name()`, die es erlaubt, diese spezielle Ergebnisrelation zu parsen.

```
$db = mysql_db_name($erg, $num);
```

Neben dem Zeiger auf das Ergebnis im Speicher muss ein Integerwert übergeben werden, der die Zeile bestimmt, die ausgelesen werden soll. In der Praxis würde das Ganze so aussehen:

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";

$dbh = mysql_pconnect($server, $user, $pass);

$erg = mysql_list_dbs($dbh);
$num = mysql_num_rows($erg);
echo "Es wurden $num Datenbanken gefunden!<br><br>";
for ($i = 0; $i < $num; $i++)
{
   $db = mysql_db_name($erg, $i);
   echo $db . "<br>";
}
?>
```

Listing 11.14: Ausgabe aller Datenbanken

Ausgabe:

```
Es wurden 7 Datenbanken gefunden!

bl
home
info_uni
mysql
name
test
xls_boerse
```

Die Funktion `mysql_list_tables()` arbeitet ähnlich wie die Funktion `mysql_list_dbs()`. Sie erwartet allerdings einen weiteren Parameter, der die Zieldatenbank festlegt, die ausgelesen werden soll.

```
$erg = mysql_list_tables($db, $dbh);
```

Die Funktion muss in Kombination mit der Auswertungsfunktion `mysql_tablename()` verwendet werden.

```
$db = mysql_tablename($erg, $i);
```

Genau wie `mysql_db_name()` liest die Funktion die Daten aus dem Speicher aus und macht sie im Programm verfügbar.

```
$erg = mysql_list_tables("name", $dbh);
$num = mysql_num_rows($erg);
echo "Es wurden $num Tabellen gefunden!<br><br>";
for ($i = 0; $i < $num; $i++)
{
   $name = mysql_tablename($erg, $i);
   echo $name . "<br>";
}
```

Ausgabe:

```
Es wurden 3 Tabellen gefunden!

geburtstag
telnum
test
```

Die letzte der drei Funktionen ist `mysql_list_fields()`, die eine Tabelle einer bestimmten Datenbank analysiert. Die Funktion erwartet drei Parameter, die darüber Auskunft geben, welche Tabelle genau analysiert werden soll.

```
$erg = mysql_list_fields($db, $table, $dbh);
```

Der erste Parameter übergibt den Namen der Zieldatenbank in Form eines Strings. Der zweite Parameter wird ebenfalls als String übergeben und bezeichnet die Tabelle, auf die die Funktion zugreifen soll. Der letzte Parameter ist das obligatorische Verbindungshandle auf eine geöffnete Datenbank.

Im Gegensatz zu den ersten zwei Funktionen stehen für das Ergebnis der Funktion `mysql_list_fields()` gleich vier Auswertungsfunktionen zur Verfügung:

- `mysql_field_name($erg, $num)`
  Gibt den Namen der Spalte als String zurück.
- `mysql_field_table($erg, $num)`
  Gibt den Namen der Tabelle als String zurück.

- `mysql_field_type($erg, $num)`
  Gibt den Wertetyp der Spalte als String zurück.
- `mysql_field_flags($erg, $num)`
  Gibt die Optionen der Spalte als String zurück.

Alle vier Funktionen erwarten dieselben Parameter: den Zeiger auf die Ergebnisrelation im Speicher, die durch die Funktion `mysql_list_fields()` erstellt wurden, und einen Integerwert, der den Zieldatensatz angibt.

```
<?
$server = "localhost";
$user = "root";
$pass = "";

$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$erg = mysql_list_fields("name", "telnum", $dbh);
$num = mysql_num_fields($erg);
echo "Es wurden $num Spalten gefunden!<br><br>";
for ($i = 0; $i < $num; $i++)
{
   $name = mysql_field_name($erg, $i);
   $type = mysql_field_type($erg, $i);
   $option = mysql_field_flags($erg, $i);
   echo $name . " (" . $type . " " . $option . ")<br>";
}
?>
```

Listing 11.15: Übersicht über die Tabelle »telnum« in der Datenbank »name«

Ausgabe:

```
Es wurden 4 Spalten gefunden!

id (int not_null unique_key auto_increment)
name (string )
vorwahl (int )
telefon (int not_null primary_key)
```

Das Skript analysiert die Tabelle »telnum« in der Datenbank »name« und gibt eine komplette Übersicht über alle Informationen der einzelnen Spalten. Diese Aufstellung entspricht der SQL-Anweisung `DESCRIBE` und liefert dieselben Ergebnisse.

11.4.4 Fehlerbehandlung mit dem MySQL-Modul

Wie Sie vielleicht schon bemerkt haben, gibt PHP im Fehlerfall bezüglich einer SQL-Anweisung keine Fehlermeldung aus. Die falsche Anfrage wird schlicht ignoriert und das Skript setzt seine Arbeit mit vermutlich sonderbaren Ergebnissen fort.

Dieses Verhalten ist allerdings nicht weiter verwunderlich, da der PHP-Interpreter nur Fehler bezüglich des eigenen PHP-Codes ausgibt. Er hat keine Möglichkeit die Korrektheit von SQL-Anfragen zu kontrollieren. Diese Aufgabe übernimmt der MySQL-Datenbankserver, der syntaktisch falsche Anweisungen mit einer Fehlermeldung abweist und zurückschickt. Diese Meldungen werden von PHP nicht einfach geschluckt, sondern in einem speziellen Speicherbereich abgelegt, wo sie mit einer bestimmten Funktion abgerufen werden kann.

Funktion	Beschreibung
`mysql_errno()`	Gibt die Nummer der letzten Fehlermeldung aus.
`mysql_error()`	Gibt den Text zur letzten Fehlermeldung aus.

Tabelle 11.30: Fehlerbehandlungsfunktionen

Jeder Fehlertyp wird in MySQL mit einer bestimmten Nummer versehen, die es ermöglicht, den Fehler intern zu identifizieren. Die Fehlernummer des letzten aufgetretenen Fehlers kann über die Funktion `mysql_errno()` abgerufen werden. Die Funktion erwartet lediglich den Verbindungshandle zur geöffneten Datenbank.

```
$errornum = mysql_errno($dbh);
```

Damit der Fehler auch als Klartext vorliegt, besteht die Möglichkeit mit der Funktion `mysql_error()` den Fehlertext der MySQL-Software auszulesen. Auch hier muss lediglich das Verbindungshandle als Parameter übergeben werden.

```
$error = mysql_error($dbh);
```

In der Praxis empfiehlt es sich, die beiden Werte zu kombinieren, um ein möglichst vollständiges Bild des Fehlers zu erhalten. Das nachfolgende Beispiel zeigt, wie diese beiden Funktionen in einem Skript eingesetzt werden können.

```
<?
$server = "localhost";
$user = "root";
$pass = "";
$db = "name";

$dbh = mysql_pconnect($server, $user, $pass);
mysql_select_db($db, $dbh);

$query = "SELECT * FROM test";
mysql_query($query, $dbh);

$errornum = mysql_errno($dbh);
$error = mysql_error($dbh);
echo $errornum . ": " . $error;
?>
```

Listing 11.16: Fehlerbehandlung

Je nachdem welcher Fehler in der SQL-Anfrage vorliegt, hat das Skript eine andere Ausgabe. Wenn die Anfrage korrekt war und erfolgreich ausgeführt wurde, wird der Fehlercode 0 zurückgegeben. Im Folgenden sehen Sie ein paar typische Beispiele mit unterschiedlichen Fehlertypen.

Syntaxfehler

```
ELECT * FROM test;
```

Ausgabe:

```
1064: You have an error in your SQL syntax near 'ELECT * FROM test'
at line 1
```

Tabelle existiert nicht

```
SELECT * FROM kunde;
```

Ausgabe:

```
1146: Table 'name.kunden' doesn't exist
```

Datenbank existiert nicht

```
USE datenbank;
```

Ausgabe:

```
1049: Unknown database 'datenbank'
```

Spalte existiert nicht

```
SELECT farbe FROM test;
```

Ausgabe:

```
1054: Unknown column 'farbe' in 'field list'
```

Tabellendefinition stimmt mit Anweisung nicht überein

```
INSERT INTO test VALUES (12, 'Hundekuchen', 'gestern', 24.5)
```

Ausgabe:

```
1136: Column count doesn't match value count at row 1
```

11.5 Einige Tipps zum Schluss

Am Ende dieses wirklich sehr langen Kapitels über Datenbanken möchte ich noch einige Tipps präsentieren, die Ihnen hoffentlich helfen, besser mit PHP und der MySQL-Datenbank umzugehen.

11.5.1 Benutzen Sie ein Konfigurationsskript

Wenn Sie mit mehreren Skripten auf einem Server arbeiten, die Datenbankverbindungen nutzen (was fast immer der Fall ist), ist es sehr nützlich, ein Konfigurationsskript zu erstellen, das alle wesentlichen Daten für eine erfolgreiche Verbindung zu einem MySQL-Server bereithält. Das Skript sollte verschiedene Variablen definieren, die alle nötigen Daten speichern:

- Host oder IP-Adresse des Datenbankservers
- Name der genutzten Datenbank
- Benutzername
- Passwort

Außerdem sollte in dem Skript immer die Verbindung zur Datenbank erstellt werden, so dass eine Variable mit dem Datenbankhandle existert.

Das Skript sollte in einer eigenen Datei auf dem Server gespeichert und per `include()` in jedes Skript importiert werden, das diese Daten benötigt. Dieses Vorgehen hat mehrere Vorteile:

1. Falls die Zugangsdaten geändert werden, muss nur eine Datei auf dem Server angepasst werden. Anderfalls müsste jedes Skript einzeln per Hand nachgebessert werden.
2. Zugangsdaten sind sensible Daten, die von anderen eingesehen werden sollten. Solange die Daten nur in einer Datei abgespeichert sind, hat man volle Kontrolle über die Zugriffe anderer User. Ansonsten würde mit jedem weiteren Skript die Übersicht verloren gehen.
3. Wenn die Konfigurationsdatei außerhalb des HTDOCS-Verzeichnisses des http-Servers liegt, besteht kaum eine Möglichkeit von außen auf die Daten zuzugreifen. Darum sollte die Datei immer in einem eigenen Verzeichnis abgespeichert werden.

11.5.2 Nutzen Sie mysql_pconnect()

Wenn es möglich ist, benutzen Sie die Funktion `mysql_pconnect()`, um die Datenbankverbindung zu erstellen. Die Funktion arbeitet schneller als `mysql_connect()`, wenn ein Skript sich über mehrere Seiten erstreckt, da nicht immer wieder aufs Neue eine Verbindung aufgebaut werden muss.

11.5.3 Vergeben Sie sinnvolle Namen

Nichts ist schlimmer als das große Raten nach einige Monaten, wenn es darum geht festzustellen, welche Tabellen in der Datenbank welche Aufgaben haben. Darum vergeben Sie immer sinnvolle Namen, die es erlauben, auch nach einiger Zeit festzustellen, was diese Tabelle speichert und ob sie noch benötigt wird. Das gilt sowohl für Datenbanken als auch für Spaltennamen, denn auch SQL-Anweisungen können zu unleserlichem Kauderwelsch werden, wenn man nur mit Kürzeln arbeitet.

Einige Provider bieten auf ihrem Server, neben den bekannten Leistungen wie Speicherplatz und E-Mail-Adressen, gegen Aufpreis die Möglichkeit Skripte mit Datenbankverbindungen auszuführen. Meistens sind die Zugriffsrechte auf dem Datenbankserver auf eine einzige Datenbank beschränkt, so dass es keine Möglichkeit gibt verschiedene Projekte auf verschiedene Datenbanken zu verteilen.

Hier hilft es, wenn man sauber strukturierte Bezeichner für die einzelnen Tabellen wählt. In der Praxis wird meistens so verfahren, dass jede Tabelle mit einem Präfix versehen wird, das Auskunft über die Zugehörigkeit gibt.

```
proj_kunden
proj_zahlungen
proj_aufträge
single_newsletter
single_content
kb_header
kb_content
kb_bga
```

Neben dem Vorteil, dass auf den ersten Blick klar ist, wo diese Tabelle einzuordnen ist, werden bei alphabetischer Sortierung alle zusammengehörenden Relationen automatisch untereinander aufgereiht.

11.5.4 Achten Sie auf die Datentypen

Es ist wichtig, darauf zu achten, welche Datentypen Sie bei einer Tabellendeklaration angeben. Überlegen Sie genau, wie viel Platz Sie brauchen und welcher Datentyp am besten für diese Aufgabe in Frage kommt. Es ist zum Beispiel wenig geschickt, Vorwahlnummern als Zahlenwerte zu speichern, da alle führenden Nullen verloren gehen (wie Sie bereits gesehen haben). Dasselbe gilt für Postleitzahlen und Handynummern.

Ein weiterer gern gemachter Fehler ist der »Trick« alle Spalten mit dem Datentyp BLOB zu versehen. Diese Technik bietet zwar den Vorteil, dass man keine Probleme mit dem Speicherplatz hat und keine Daten verloren gehen. Das Problem dabei ist aber, dass BLOB-Daten von MySQL automatisch in Textdateien ausgelagert werden, damit genug Speicherplatz zur Verfügung steht. Es ist klar, dass die Performance darunter leidet, wenn MySQL bei jedem Zugriff auf externe Daten zurückgreifen muss.

11.5.5 Sparen Sie sich nicht arm

Auch auf die Gefahr, dass dieser Abschnitt deplatziert wirkt, nachdem ich auf der vorhergehenden Seite zur Sparsamkeit aufgerufen habe, sollte Sie darauf achten, sich nicht zu sehr einzuschränken. Nichts ist ärgerlicher als festzustellen, dass alle E-Mail-Adressen der Newsletterempfänger angeschnitten wurden, nur weil Sie zu kurze Datentypen angegeben haben. Das gilt selbstverständlich auch für alle anderen Daten, die von Usern eingegeben werden.

Sie sollten immer damit rechnen, dass jemand einen Doppelnamen mit 60 Zeichen hat oder eine E-Mail-Adresse angibt, die etwa so aussieht:

```
gruenesmonstervomanderenstern@raumschiff-enterprise.de
```

Darum das Resümee: Seien Sie sparsam mit den Datentypen, die Sie selber setzen, aber geben Sie den Usern ausreichend Platz für ihre Eingaben.

11.6 Übungen

1. Was ist der Unterschied zwischen den Funktionen `mysql_connect()` und `mysql_pconnect()`?
2. Was bedeutet deprecated?
3. Was ist eine Ergebnisrelation?
4. Gibt die Funktion `mysql_fetch_array()` ein nummerisches oder ein assoziatives Array zurück?
5. Was macht die Funktion `mysql_list_fields()` ?
6. Was ist der Unterschied zwischen den Funktionen `mysql_error()` und `mysql_errno()`?
7. Warum sollten Sie ein Konfigurationsskript benutzen?

12 Dynamische Grafiken mit PHP

Viele Programmierer wissen gar nicht, dass PHP weit mehr als nur dynamische HTML-Seiten erstellen kann. Eine Reihe von Erweiterungen und Modulen erlauben es, unterschiedliche Grafikformate zu erstellen und als Bild an den Browser zu schicken.

In diesem und den folgenden Kapiteln werden Sie lernen, wie man mit einfachen Mitteln Bilder und Grafiken dynamisch erzeugt und auf dem Browser ausgeben kann. Alle Programme, die ich in diesem Kapitel vorstelle, verwenden die Grafikbibliothek GD, die für PHP 4 nachträglich installiert werden muss.

Für die Erstellung von dynamischen Grafiken mit PHP benötigen Sie das optionale Grafikmodul GD für PHP 4.0. Diese Software ist nicht in der Standarddistribution der aktuellen PHP-Version vorhanden. Fragen Sie Ihren Systemadministrator, ob die GD auf Ihrem Server installiert ist oder ob er bereit ist dieses Paket nachträglich zu installieren.

Möchten Sie das Modul lokal nutzen, können Sie es unter der folgenden Adresse kostenlos downloaden:

http://www.boutell.com/gd/

Kopieren Sie die Datei `php_gd.dll` in das Systemverzeichnis Ihres Windowsbetriebssystems und aktualisieren Sie die Konfigurationsdatei php.ini. Die Datei muss die folgenden zwei Einträge beinhalten:

```
extension_dir=c:\windows\system
extension=php_gd.dll
```

Das Verzeichnis, in dem PHP-Interpreter nach den Erweiterungsmodulen suchen soll, muss an den Systempfad Ihres Rechners angepasst werden. Unter Windows NT oder Windows 2000 heißt der Systempfad standardmäßig

```
c:\winnt\system32
```

Nachdem Sie alle nötigen Einstellungen vorgenommen haben, muss der Apache Server neu gestartet werden. Danach sollten Ihnen alle Features der GD-Bibliothek zur Verfügung stehen.

12.1 Kapitelübersicht

- Das Grafikmodul GD
- Dynamische Grafiken erstellen
- Dynamische Grafiken mit HTML

12.2 Das Grafikmodul GD

Die GD-Bibliothek von Thomas Boutell erlaubt es, mit PHP dynamische Grafiken zu erstellen und auf Webseiten zu verwenden. Die Software ist aktuell in der Version 1.8.4 verfügbar und kann auf der Webseite *http://www.boutell.com/gd/* heruntergeladen werden. Eine Betaversion 2.0.1 steht ebenfalls zur Verfügung und kann zu Testzwecken heruntergeladen werden.

Die GD-Bibliothek unterstützt die Grafikformate JPG und PNG und ist in der Lage die Größe von Shockwave Flashdateien zu ermitteln. Ältere Versionen der Software waren auch in der Lage Grafiken im GIF-Format zu erstellen, doch diese Möglichkeit musste aus rechtlichen Gründen unterbunden werden. Laut den Angaben auf der Seite von Thomas Boutell liegt das Patent für das Komprimierverfahren für GIF-Dateien bei der Firma Unisys. Die Verwendung dieser Technik wäre ein Copyrightbruch und wurde deshalb aus der GD-Bibliothek entfernt.

Alle Beispiele für dynamische Grafiken mit PHP beziehen sich auf das JPG-Format, das nach wie vor unterstützt wird.

12.3 Dynamische Grafiken erstellen

Während einer HTTP-Anfrage werden eine Reihe von Informationen zwischen dem Browser und dem Server ausgetauscht, die für den Datentransfer nötig sind. Jede Datei auf dem Server wird mit so genannten Header-Informationen versehen, die unter anderem den MIME-Type der entsprechenden Datei speichern. So ist der Browser in der Lage die empfangene Datei richtig zu interpretieren und ein Bild von einer einfachen Textdatei zu unterscheiden.

Die folgende Tabelle gibt einen kurzen Überblick über die häufigsten MIME-Types im Internet:

MIME-Type	Beschreibung
`text/plain`	einfache Textdatei
`text/html`	HTML-Dokument
`image/jpeg`	Bild im jpeg-Format
`image/gif`	Bild im gif-Format
`video/mpeg`	Videodatei
`multipart/formdata`	Formulardaten mit unterschiedlichem Inhalt (Dateiupload)
`application/pdf`	PDF-Datei
`application/zip`	ZIP-Datei
`application/msword`	Word-Dokument

Tabelle 12.1: MIME-Types

PHP versieht alle Ausgaben des Interpreters automatisch mit dem MIME-Type text/html, so dass der Browser die empfangenen Daten als HTML-Dokument anzeigt. Um mit PHP Bilder erzeugen zu können, muss dieser Header überschrieben werden, damit der Browser die empfangenen Daten richtig interpretiert. Dies geschieht mit der Funktion `header()`, der als Argument der neue MIME-Type übergeben wird.

```
header("Content-type: image/jpeg");
```

Einige Browser erkennen automatisch, dass es sich nicht um HTML-Code handelt, und zeigen das erzeugte Bild auch ohne die Funktion `header()` korrekt an. Trotzdem sollte man nicht auf die Funktion verzichten, da man sich nicht immer auf diese »Intelligenz« verlassen kann.

12.3.1 Das erste Bild

Für die ersten Schritte mit der GD-Grafikbibliothek reichen eine Handvoll Funktionen, mit denen die ersten Ergebnisse erzielt werden können.

Funktion	Beschreibung
`imagecreate()`	Erschafft eine neue Grafik und gibt ein Handle auf diese zurück.
`imagerectangle()`	Zeichnet ein Rechteck in ein bestehendes Bild.
`imagecolorallocate()`	Definiert ein Farbe für ein Bild.
`imagefill()`	Füllt einen Teilbereich des Bildes mit einer bestimmten Farbe.
`imageline()`	Zeichnet ein Linie in ein Bild.
`imagesetpixel()`	Setzt einen Punkt in ein Bild.
`imagejpeg()`	Gibt ein Bild im JPG-Format aus.
`imagepng()`	Gibt ein Bild im PNG-Format aus.
`imagedestroy()`	Löscht ein Grafikhandle und gibt den Speicher frei.

Tabelle 12.2: Grafikfunktionen

Ein dynamisches Bild wird über die Funktion `imagecreate()` erschaffen, die bei erfolgreicher Ausführung ein Handle auf die neue Grafik zurückgibt. Über zwei Parameter wird die Größe des Bildes in Pixeln angegeben, die danach für weitere Funktionen zu Verfügung stehen.

```
$image = imagecreate($size_x, $size_y);
```

Ähnlich wie ein Dateihandle erlaubt das Bildhandle den Zugriff auf die neue Grafik mit weiteren Funktionen aus der GD-Bibliothek. Die leere Bildfläche kann wie in einem Malprogramm mit einer großen Anzahl von Funktionen bearbeitet werden, bevor das Ergebnis letztendlich ausgegeben wird. Ein Bild entsteht also nicht sofort, sondern wird Schritt für Schritt aufgebaut.

Die Ausgabe einer neuen Grafik geschieht über die Funktionen `imagejpeg()` oder `imagepng()`. Je nachdem welche Funktion gewählt wird, gibt das PHP-Skript die vorher definierte Bilddatei im entsprechenden Format aus. Achten Sie darauf, dass das gewählte Format mit den Angaben in der Funktion `header()` übereinstimmt.

```
imagejpeg($image);
imagedestroy($image);
```

Im letzten Schritt sollte das vorhandene Grafikhandle gelöscht werden, damit der belegte Speicherplatz wieder freigeben wird. Diese Aufgabe erledigt die Funktion `imagedestroy()`. Die Freigabe des Handles ist nicht unbedingt notwendig, wenn das Programm im nächsten Schritt beendet wird. PHP gibt automatisch den genutzten Speicher wieder frei.

Das folgende Skript zeigt den Einsatz der drei Funktionen in der Praxis:

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(150,150);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.1: Die erste Grafik

Das Ergebnis dieser paar Zeilen Code ist ein schwarzes Viereck im Browser von der Größe 150 auf 150 Pixel. Ein Rechtsklick mit der Maus auf die Eigenschaften der Grafik beweist, dass es sich tatsächlich um eine Bild im JPG-Format handelt.

Soll das Bild anders aussehen, dann muss zwischen dem Erstellen und dem Zerstören der Grafik noch etwas geschehen. Der erste Schritt ist die Definition einer neuen Farbe für die Grafik, um das bestehende Schwarz übermalen zu können. Mit der Funktion `imagecolorallocate()` kann ein neuer Farbton erschaffen werden. Dazu greift PHP auf das RGB-Farbmodell zurück und erstellt die neue Farbe anhand von drei Parametern, die Werte zwischen 0 und 255 übergeben.

```
$farbe = imagecolorallocate($image, $rot,$gruen,$blau);
```

Der erste Parameter ist das Handle auf die Grafik, für die der Farbton gelten soll. Die drei folgenden Parameter müssen Integerwerte zwischen 0 und 255 sein, die den jeweiligen Farbanteil des neuen Farbtons bestimmen. Auf diese Weise können über 16.5 Millionen (= 255^3) Farben definiert werden, sofern die vorhandene Grafikkarte diesen Farbvielfalt überhaupt unterstützt.

Viele Zeichenfunktionen der GD-Grafikbibliothek verlangen als Parameter den Verweis auf eine Farbe. Die Funktion `imagefill()` braucht diesen Parameter beispielsweise um festzulegen, in welchem Farbton ein Bereich der Grafik gefüllt werden soll. Dabei arbeitet die Funktion ähnlich wie ein Füllwerkzeug in einem Grafikprogramm: Sie füllt die komplette Fläche einer Grafik, bis sie auf eine farbliche Grenze stößt. Existiert diese Grenze nicht, dann wird das komplette Bild übermalt.

```
imagefill($image, $x, $y, $farbe);
```

Ähnlich wie bei `imagefill()` verhält es sich auch bei anderen Grafikfunktionen wie zum Beispiel `imageline()`, `imagerectangle()` oder `imagesetpixel()`. Alle drei Funktionen erwarten als letzten Parameter eine Farbangabe, die festlegt, in welcher Farbe das gewünschte geometrische Objekt gezeichnet werden soll. Die Funktionsnamen sind alle intuitiv angelegt und so wird schnell deutlich, welche Aufgaben mit den Funktionen bewältigt werden können.

```
imagerectangle($image, $x1, $y1, $x2, $y2, $farbe);
// zeichnet ein Rechteck
imageline($image, $x1, $y1, $x2, $y2, $farbe);
// zeichnet eine Gerade
imagesetpixel($image, $x, $y, $farbe);
// zeichnet einen Punkt (Pixel)
```

Der Aufbau der Funktionen ist im Prinzip immer gleich und variiert nur in der Anzahl der Parameter. Voraussetzung für jede Grafikfunktion ist ein gültiges Handle auf eine Grafik und definierte Farbe.

Das folgende Skript zeigt diese Funktionen in der Praxis und vermittelt ein Gefühl für den Einsatz der Grafikbibliothek von PHP. Es soll eine einfache mathematische Funktion berechnet werden, die anschließend als Grafik im Browser erscheint.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(400,200);
//Farben werden definiert
$grau = imagecolorallocate($image, 200, 200, 200);
$schwarz = imagecolorallocate($image, 0, 0, 0);
$blau = imagecolorallocate($image, 0, 0, 255);
//Bild wird grau übermalt und mit Rahmen versehen
imagefill($image, 0, 0, $grau);
imagerectangle($image, 0, 0, 399, 199, $schwarz);
//waagerechte Linie
imageline($image, 0, 100, 400, 100, $schwarz);
//Sinusfunktion
for($i=-200; $i<400; $i++)
{
        imagesetpixel($image, $i, (sin($i/30)*90)+100, $blau);
}
//Bild wird ausgegeben
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.2: Grafische Darstellung einer Sinusfunktion

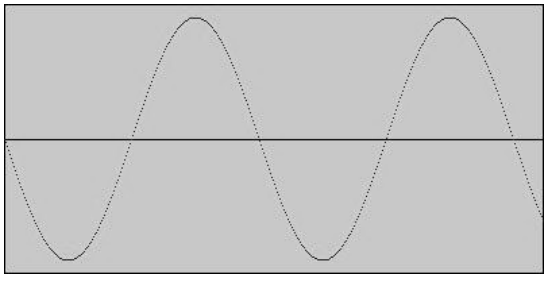


Abbildung 12.1: Die Ausgabe der Sinusfunktion

Das Skript zeichnet ein Koordinatensystem mit einer Sinuskurve in den Browser. Realisiert wird die Grafik durch die mathematische Funktion `sin()`, die Sie bereits am Anfang des Buches kennen gelernt haben. Für jeden Punkt auf der X-Achse wird durch die Funktion der entsprechende Punkt auf der Y-Achse berechnet, der dann über die Funktion `imagesetpixel()` in die Grafik gezeichnet wird.

12.3.2 Weitere Grafikfunktionen

Das Grafikmodul GD bietet eine ganze Reihe von Funktionen, die wir bisher noch nicht besprochen haben. Dieses Kapitel soll einen kurzen Überblick über die bestehenden Möglichkeiten geben und eine Reihe von wichtigen Funktionen vorstellen. Alle Funktionen werden im Laufe des Kapitels mit einem Beispiel erklärt.

Funktion	Beschreibung
`getimagesize()`	Gibt ein Array mit Informationen über ein GIF-, JPG- oder PNG-Bild zurück.
`imagearc()`	Zeichnet eine unvollständige Ellipse.
`imagecopy()`	Kopiert einen Teil eines Bildes.
`imagecopyresized()`	Kopiert einen Teil eines Bildes und verändert die Größe.
`imagedashedline()`	Zeichnet eine gestrichelte Linie.
`imagefilledpolygon()`	Zeichnet ein gefülltes Vieleck (Polygon).
`imagefilledrectangle()`	Zeichnet ein gefülltes Rechteck.

Tabelle 12.3: Weitere Grafikfunktionen

Funktion	Beschreibung
imagepolygon()	Zeichnet ein Vieleck (Poligon)
imagestring()	Zeichnet eine Zeichenkette horizontal.
imagestringup()	Zeichnet eine Zeichenkette vertikal.
imagesx()	Ermittelt die Breite eines Bildes.
imagesy()	Ermittelt die Höhe eines Bildes.
imagecolorat()	Ermittelt den Farbwert eines Punktes.

Tabelle 12.3: Weitere Grafikfunktionen (Forts.)

getimagesize()

Neben der Möglichkeit eigene Bilder zu erschaffen, liefert die GD-Bibliothek auch Funktionen, um Informationen über schon bestehende Bilder auszulesen. Die Funktion `getimagesize()` gibt ein Array mit Informationen über den Typ und über die Größe eines Bildes zurück. Als Beispielbild verwende ich die Datei img.jpg.

```
<?
$bild = "img.jpg";
$info = getimagesize($bild);
echo "Breite des Bildes: $info[0]<br>";
echo "Höhe des Bildes: $info[1]<br>";
echo "Grafikformat: $info[2]<br><br>";
echo "<img src=$bild $info[3]>";
?>
```

Listing 12.3: Informationen über ein Bild

Die Funktion `getimagesize()` gibt ein nummerisches Array mit vier Elementen zurück. Die einzelnen Indizes haben folgende Bedeutung:

- 0: Breite des Bildes in Pixel
- 1: Höhe des Bildes in Pixel
- 2: Grafikformat: 1 = GIF, 2 = JPG, 3 = PNG
- 3: Zeichenkette für die Einbindung in den IMG-Tag

```
width="xxx" height="xxx"
```

Abbildung 12.2: Skriptausgabe mit dem HTML-Quelltext

Das Skript gibt die einzelnen Werte des Bildes aus und nutzt den letzten Index des Arrays für die Ausgabe des Bildes mit exakten Größenangaben. Das Bild zeigt das Ergebnis des Skripts sowie den HTML-Quellcode der Webseite.

imagearc()

Die Funktion `imagearc()` zeichnet Ellipsen und Teilbereiche von Ellipsen in eine Grafik.

```
imagearc($image, $x, $y, $width, $height, $winkel1, $winkel2,
$farbe);
```

Der erste Parameter ist inzwischen bekannt und gibt über ein Handle die Zielgrafik an die Funktion weiter. Die Werte `$x` und `$y` legen das Zentrum der Ellipse fest, wobei der Punkt 0,0 die obere linke Ecke der Grafik ist. Die Parameter `$width` und `$height` bestimmen die Höhe und Breite der Ellipse und erlauben so festzulegen, ob das Ergebnis tatsächlich eine Ellipse ist oder doch ein geometrischer Kreis. Die letzten beiden Parameter vor der Festlegung der Farbe geben den Start- und Zielwert der Winkel an, zwischen denen die Ellipse gezeichnet werden soll.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(300,100);
$schwarz = imagecolorallocate($image, 0, 0, 0);
$grau = imagecolorallocate($image, 200, 200, 200);
imagefill($image, 0, 0, $grau);
imagearc($image, 50, 50, 60, 80, 0, 300, $schwarz);
imagearc($image, 150, 50, 80, 30, 90, 270, $schwarz);
imagearc($image, 250, 50, 90, 90, 0, 360, $schwarz);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.4: Die Funktion imagearc()

Abbildung 12.3: Drei Ellipsen mit Lücken

imagedashedline()

Die Funktion `imagedashedline()` ist eine sehr einfache Funktion und arbeitet genauso wie `imageline()`.

```
imagedashedline($image, $x1, $y1, $x2, $y2, $farbe);
```

Neben den Imagehandle und der Farbe werden die Start- und Zielkoordinaten für die Punkte angegeben, zwischen denen die gestrichelte Linie verlaufen soll. Der Abstand zwischen den Strichen kann nicht beeinflusst werden.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(200,20);
$schwarz = imagecolorallocate($image, 0, 0, 0);
$grau = imagecolorallocate($image, 200, 200, 200);
imagefill($image, 0, 0, $grau);
imagedashedline($image, 10, 10, 190, 10, $schwarz);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.5: Die Funktion imagedashedline()

Abbildung 12.4: Die Linie

imagefilledrectangle()

Die Funktion `imagerectangle()` haben Sie bereits kennen gelernt. Die neue Funktion `imagefilledrectangle()` zeichnet ebenfalls ein Rechteck in ein Bild, mit dem Unterschied, dass das neue Rechteck ausgefüllt ist.

```
imagefilledrectangle($image, $x1, $y1, $x2, $y2, $farbe);
```

Genau wie bei der Schwesterfunktion `imagerectangle()` werden neben Bildhandle und der Farbe zwei Koordinaten übergeben, die die obere linke und die untere rechte Ecke des Rechtecks bestimmen.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(200,200);
$schwarz = imagecolorallocate($image, 0, 0, 0);
$blau = imagecolorallocate($image, 0, 0, 255);
imagefilledrectangle($image, 0, 0, 200, 200, $blau);
imagefilledrectangle($image, 30, 30, 170, 170, $schwarz);
imagefilledrectangle($image, 60, 60, 140, 140, $blau);
imagefilledrectangle($image, 90, 90, 110, 110, $schwarz);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.6: Die Funktion imagefilledrectangle()

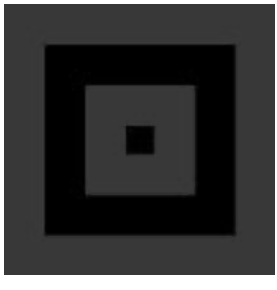


Abbildung 12.5: Vier gefüllte Rechtecke

Das Skript zeichnet vier gefüllte Rechtecke abwechselnd in den Farben Blau und Schwarz übereinander. Da die Zeichnungen immer kleiner werden, ergibt sich ein symmetrisches Muster.

imagepolygon() und imagefilledpolygon()

Mit den Funktionen `imagepolygon()` und `imagefilledpolygon` können beliebige Vielecke gezeichnet werden, die je nach Funktion ausgefüllt sind oder nicht. Die Eckpunkte für das zu zeichnende Polygon werden über ein Array an die Funktion weitergegeben. Der Aufbau des Array muss sich dabei an folgende Struktur halten:

```
punkte[0] = x1
punkte[1] = y1
punkte[2] = x2
punkte[2] = y2
...
```

Neben dem Array und den üblichen Angaben zu Farbe und Zielbild muss noch die Anzahl der Punkte als Parameter übergeben werden.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(300,150);
$rot = imagecolorallocate($image, 255, 0, 0);
$weiss = imagecolorallocate($image, 255, 255, 255);
imagefilledrectangle($image, 0, 0, 300, 150, $weiss);
$punkte1 = array(75,7,  125,125,  10,55,  140,55,  20,120);
$punkte2 = array(225,7,  275,125,  160,55,  290,55,  170,120);
imagepolygon($image, $punkte1, count($punkte1)/2, $rot);
imagefilledpolygon($image, $punkte2, count($punkte1)/2, $rot);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.7: Die Funktionen imagepolygon() und imagefilledpolygon()

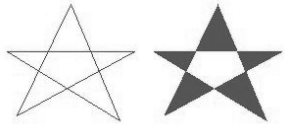


Abbildung 12.6: Zwei Sterne

Beide Funktionen schließen das Polygon automatisch ab und verbinden den letzten Punkt des Arrays mit dem Startpunkt der Zeichnung. Die Funktion `imagefilledpolygon()` füllt dabei das Innere der Zeichnung mit der angegebenen Farbe, was bei sich überschneidenden Linien auf den ersten Blick etwas verwirrend aussieht.

Jede Linie hat eine Innen- und eine Außenseite, die durch den Startpunkt definiert werden. Die Funktion füllt nicht die komplette Grafik aus, weil der innere Bereiche wieder durch überschneidende Linien komplett abgegrenzt wird und dadurch (dank der PHP-Logik) nicht zur eigentlichen Zeichnung gehört. Will man den Stern komplett rot erscheinen lassen, muss eine weitere Funktion bemüht werden.

imagestring() und imagestringup()

Die Funktionen `imagestring()` und `imagestringup()` dienen zur Ausgabe von Texten innerhalb einer Grafik. Der String kann pixelgenau gesetzt werden und beginnt immer an der oberen linken Ecke der x- und y-Angabe in der Funktion. Mit der Funktion `imagestringup()` ist es möglich, Texte horizontal in eine Grafik zu »schreiben«.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreate(200,100);
$black = imagecolorallocate($image, 0, 0, 0);
$weiss = imagecolorallocate($image, 255, 255, 255);

imagefilledrectangle($image, 0, 0, 300, 150, $weiss);
for($i=1; $i<6; $i++)
{
imagestring($image, $i, 10, $i*12, "Hallo PHP!", $black);
imagestringup($image, $i, ($i*11)+100, 90 , "Hallo PHP!", $black);
}
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.8: Texte in Grafiken

Abbildung 12.7: Text in alle Richtungen

Neben dem Grafikhandle und den Koordinaten in der Funktion muss noch festgelegt werden, welche Größe die Schrift haben soll. Die Angabe erfolgt direkt nach dem Handle auf die Grafik und muss sich zwischen 1 und 5 bewegen.

12.3.3 Bilder einlesen und ausgeben

Eine interessante Möglichkeit der GD-Grafikbibliothek ist das Einlesen von schon vorhandenen Bildern, um sie mit den bestehenden Grafikfunktionen weiter zu verarbeiten. Dabei ist es durchaus möglich, das Format der Grafik zu ändern und beispielsweise eine PNG-Grafik als JPEG-Datei wieder auszugeben.

Funktion	Beschreibung
`imagecreatefromjpg()`	Erzeugt ein Bild auf der Basis einer JPEG-Datei.
`imagecreatefrompng()`	Erzeugt ein Bild auf der Basis einer PNG-Datei
`imagecreatefromgif()`	Erzeugt ein Bild auf der Basis einer GIF-Datei

Tabelle 12.4: Funktionen zur Bildbearbeitung

Das Vorgehen zur Erstellung eines Grafikhandles aus einem bestehenden Bild unterscheidet sich kaum von der bisherigen Methode. Die Funktion `imagecreate()` wird einfach durch eine der Funktionen aus der obigen Tabelle ersetzt, die je nach Grafikvorlage benutzt werden müssen. Als Parameter wird der Pfad oder die URL zur entsprechenden Grafikdatei angegeben.

```
$image = imagecreatefromjpeg($djpgDatei);
```

Die Funktion gibt wie gehabt ein Grafikhandle zurück, mit dem weiter gearbeitet werden kann. Der einfachste Schritt ist natürlich, das Bild gleich wieder auszugeben.

```
<?
header( "Content-type: image/png");
$image = imagecreatefromjpeg("img.jpg");
imagepng($image);
imagedestroy($image);
?>
```

Listing 12.9: Umwandlung von JPG in PNG

Das kurze Skript liest eine JPEG-Datei ein und gibt sie im nächsten Schritt wieder als PNG-Datei aus. Die Ausgabe eines Bildes ist sicherlich einfacher zu bewerkstelligen, aber die implizite Umwandlung mit dem GD-Modul kann sich durchaus sehen lassen.

Abbildung 12.8: Der Chiemsee als PNG-Datei

Der Einsatz der übrigen Funktionen ist für alle Grafikformate identisch, so dass es nicht nötig ist, zwei weitere Beispiele für `imagecreatefromgif()` und `imagecreatefrompng()` zu geben.

Umso interessanter ist dafür die Möglichkeit bestehende Grafiken mit dem GD-Modul dynamisch zu bearbeiten und mit eigenen Zeichnungen zu erweitern. Das folgende Beispiel beschreibt einen Schritt in diese Richtung.

```
<?
header( "Content-type: image/jpeg");
$image = imagecreatefromjpeg("img.jpg");
$schwarz = imagecolorallocate($image, 255, 255, 255);
imagestring($image, 5, 30, 30, "CHIEMSEE", $schwarz);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.10: Chiemsee mit Schrift

Abbildung 12.9: JPEG-Grafik mit Aufschrift

Das Skript liest das Bild img.jpg ein und platziert den Text CHIEMSEE in die Mitte der Grafik. Für die Ausgabe wurde die Funktion `imagestring()` verwendet.

12.4 Dynamische Grafiken mit HTML

Die bisherigen Kapitel haben den Einsatz der GD-Grafikbibliothek erklärt, ohne weiter auf den Einsatz in kompletten HTML-Seiten einzugehen. Da ein PHP-Skript, das eine dynamische Grafik generiert, keine HTML-Formatierung enthalten kann, muss die eigentliche Seite in einer eigenen Datei gespeichert sein. Der Aufruf des dynamischen Bildes geschieht über den bekannten HTML-Tag <IMG ...>.

Als Bildquelle wird einfach das PHP-Skript angegeben, das für die Erstellung der Grafik zuständig ist. Eventuelle Parameter können wie gewohnt über die GET-Methode übergeben werden. Für die kommenden Beispiele wird das folgende PHP-Skript als Grundlage verwendet:

```
<?
header( "Content-type: image/jpeg");
if(!(isset($data))) {$data="nobody";}
if(!(isset($f1))) {$f1=255;}
if(!(isset($f2))) {$f2=255;}
if(!(isset($f3))) {$f3=255;}
$image = imagecreate(300,120);
$black = imagecolorallocate($image, 0, 0, 0);
$colr = imagecolorallocate($image, $f1, $f2, $f3);
imagefilledrectangle($image, 0, 0, 300, 120, $colr);
imagerectangle($image, 0, 0, 299, 119, $black);
imagestring($image, 5, 85, 40, "Hallo $data !", $black);
imagestring($image, 4, 20, 60, "Das ist deine persönliche Grafik!",
$black);
imagejpeg($image);
imagedestroy($image);
?>
```

Listing 12.11: Dynamische Bilderzeugung über eine HTML-Seite

Das Skript erstellt eine Art Visitenkarte im JPEG-Format. Es ist möglich, über verschiedene Parameter den Namen auf der Karte sowie die Hintergrundfarbe zu beeinflussen. Standardmäßig ist die Karte weiß.

Das folgende HTML-File zeigt den Einsatz der dynamischen Grafiken auf einer Internetseite. Das Aussehen der Bilder wird über die vorgegebenen Parameter gesteuert. Die Datei selber ist kein Skript, sondern eine einfache Webpage.

```
<HTML>
<HEAD><TITLE>Visitenkarten</TITLE>
<BODY>
<TABLE WIDTH="600" BORDER=0>
<TR><TD>
<IMG SRC="listing143.php?data=Dagobert&f1=255&f2=100&f3=100" BOR-
DER=0>
Eine rote Karte für Dagobert!
</TD><TD>
<IMG SRC="listing143.php?data=June&f3=0" BORDER=0>
Eine gelbe Karte für June!
</TD></TR><TR><TD>
<IMG SRC="listing143.php?f1=100&f2=255&f3=10" BORDER=0>
Eine hellgrüne Karte für niemand!
</TD><TD>
<IMG SRC="listing143.php?data=Dirk" BORDER=0>
Eine weisse Karte für mich!
</TD></TR></TABLE>
</BODY>
</HTML>
```

Listing 12.12: HTML-Datei mit dynamischen Bildern

Abbildung 12.10: Bunte Karten, die s/w leider nicht so beeindruckend sind.

Der Trick beim Einsatz dynamischer Bilder auf HTML-formatierten Seiten liegt ganz einfach in den ausgelagerten Skriptdateien, die über Parameter gesteuert werden. Jede Grafik kann beliebig oft über den <IMG ...>-Tag eingebaut werden, da der Browser das Skript als einfache Grafik interpretiert. Die HTML-Codierung kann selbstverständlich auch durch ein weiteres PHP-Skript erzeugt worden sein, da die Bilder erst vom Clientrechner aus geladen werden können.

12.5 Übungen

Ü

1. Was ist die Voraussetzung für die Erstellung von dynamischen Grafiken?
2. Was gibt die Funktion `imagecreate()` zurück?
3. Zeichnet die Funktion `imagearc()` Kreise oder Ellipsen?
4. Was unterscheidet die Funktion `imagerectangle()` von der Funktion `imagefilledrectangle()`?
5. Wie können dynamische Grafiken in eine HTML-Datei gebracht werden?

13 Das File Transfer Protocol

Das FTP-Protokoll dient zum Austausch von Dateien zwischen verschiedenen Rechnern in einem Netzwerk. Im Gegensatz zum HTTP-Protokoll, das für den Datentransfer im World Wide Web eingesetzt wird, erlaubt das FTP-Protokoll auch die Manipulation von Daten auf fremden Rechnern. So ist es beispielsweise möglich, mit den entsprechenden Zugriffsrechten Daten auf einen Webserver zu kopieren, der auf der anderen Seite der Welt steht.

Das Kürzel FTP steht für die Bezeichnung File Transfer Protokoll, was übersetzt soviel bedeutet wie *Datei Transfer Konventionen*. Die Übersetzung klingt etwas steif, trifft aber meiner Meinung nach den Kern der Sache: Ein Protokoll legt bestimmte Konventionen fest, an die sich beide Seiten einer Rechnerverbindung halten müssen, wenn sie erfolgreich kommunizieren wollen. Im Fall des FTP-Protokolls wird über diese Angaben geklärt, wie der Datenaustausch zwischen den Servern stattfinden soll. Zu diesem Zweck gibt es eine ganze Reihe so genannter FTP-Befehle, mit denen manuell Dateien eines Rechners abgerufen bzw. hochgeladen werden können.

Glücklicherweise sind diese Befehle in den meisten FTP-Programmen heute so untergebracht, dass der Anwender davon kaum noch etwas mitbekommt. Da alle PHP-Funktionen bezüglich des FTP-Protokolls auf derselben Ebene arbeiten, werde ich hier nicht weiter auf den FTP-Befehlssatz eingehen.

13.1 Kapitelübersicht

- FTP mit PHP
- Die erste Serververbindung
- Navigation auf dem Server
- Datentransfer und Manipulation von Serverdaten

13.2 FTP mit PHP

Mit den FTP-Funktionen des PHP-Interpreters kann auf einen fremden Server zugegriffen werden, sofern dieser das Protokoll unterstützt. Weiterhin sind natürlich die entsprechenden Zugangsdaten notwendig, um über FTP arbeiten zu können. PHP erlaubt es nicht, einen eigenen FTP-Service auf einem Server zu errichten.

Alle FTP-Funktionen beginnen mit dem Kürzel ftp gefolgt von einem Unterstrich. Die folgende Tabelle zeigt die wichtigsten Funktionen für den Umgang mit dem FTP-Protokoll.

Funktion	Beschreibung
`ftp_connect()`	Verbindet mit einem FTP-Server.
`ftp_login()`	Überträgt die Nutzerdaten an den Server.
`ftp_pwd()`	Gibt den Namen des aktuellen Verzeichnisses zurück.
`ftp_chdir()`	Wechselt das Verzeichnis auf dem FTP-Server.
`ftp_cdup()`	Geht eine Verzeichnisebene höher auf dem FTP-Server.
`ftp_mkdir()`	Erstellt ein Verzeichnis auf dem FTP-Server.
`ftp_rmdir()`	Löscht ein Verzeichnis auf dem FTP-Server.
`ftp_nlist()`	Erstellt eine Dateiliste eines Verzeichnisses.
`ftp_rawlist()`	Erstellt eine detaillierte Dateiliste eines Verzeichnisses.
`ftp_systype()`	Gibt das Betriebssystem des FTP-Servers zurück.
`ftp_get()`	Lädt eine Datei vom FTP-Server herunter.
`ftp_put()`	Lädt eine Datei auf den FTP-Server.
`ftp_size()`	Gibt die Größe einer Datei in Byte zurück.
`ftp_mdtm()`	Gibt das Datum der letzten Änderung einer Datei zurück.
`ftp_rename()`	Benennt eine Datei auf dem Server um.
`ftp_delete()`	Löscht eine Datei auf dem Server.
`ftp_quit()`	Trennt die FTP-Verbindung.

Tabelle 13.1: FTP-Funktionen unter PHP

13.3 Die erste Serververbindung

Bevor Daten mit einem fremden Rechner ausgetauscht werden können, muss eine Verbindung hergestellt werden. Das geschieht unter PHP in zwei Schritten: Als Erstes wird über die Funktion `ftp_connect()` der Zielserver kontaktiert.

```
$handle = ftp_connect($hostname);
```

Ist die Verbindung erfolgreich, gibt die Funktion ein FTP-Handle zurück, das für alle weiteren Funktionen benötigt wird. Über einen optionalen zweiten Parameter kann der Port für die FTP-Verbindung festgelegt werden. Wird der Port nicht angegeben, verwendet PHP automatisch den Standardport 21. Im zweiten Schritt muss man sich gegenüber dem kontaktierten Server über die Funktion `ftp_login()`, die neben dem FTP-Handle den Usernamen und das Passwort übergibt, identifizieren.

```
$con = ftp_login($handle, $user, $pwd);
if ($con) {echo "Anmeldung erfolgreich!";}
```

Ist die Anmeldung erfolgreich, gibt die Funktion TRUE zurück. Das Ergebnis sollte auf jeden Fall überprüft werden, bevor im Programm fortgefahren wird. Alle weiteren FTP-Funktionen benötigen diese Eröffnung der Verbindung, um arbeiten zu können.

Wenn eine FTP-Verbindung nicht mehr benötigt wird, sollte sie wieder geschlossen werden. Dieser Schritt ist weit weniger kompliziert und passiert über die Funktion `ftp_quit()`. Als Parameter wird das Handle der nicht mehr benötigten Connection übergeben.

```
ftp_quit($handle);
```

Die Verbindung wird geschlossen und das Handle wieder freigegeben.

```
<?
$hostname = "www.kulturbrand.de";
$user = "root";
$pwd = "geheim";
$handle = ftp_connect($hostname);
$con = ftp_login($handle, $user, $pwd);
if ($con) {echo "Anmeldung erfolgreich!";}
ftp_quit($handle);
?>
```

Listing 13.1: Erster Kontakt mit einem FTP-Server

Ausgabe:

```
Anmeldung erfolgreich!
```

13.4 Navigation auf dem Server

Nach der erfolgreichen Anmeldung an einem Server steht dem Zugriff auf die vorhandenen Daten bzw. dem Upload neuer Daten fast nichts mehr im Wege. Allerdings ist es oft nötig, sich zuerst durch das Verzeichnissystem des Servers zu navigieren, um den Zielbereich für den Zugriff zu finden. Für diese Aufgabe stellt PHP einige Funktionen zur Verfügung, die hier der Reihe nach vorgestellt werden.

Für die folgenden Programmfragmente wird immer eine bestehende FTP-Connection vorausgesetzt, die mit Programmcode aus dem letzten Kapitel erstellt wurde.

Die Funktion ftp_pwd()

Die Funktion `ftp_pwd()` ist sehr einfach gestrickt und gibt schlicht den Namen des akuellen Verzeichnisses zurück, auf das ein FTP-Handle zeigt.

```
$name = ftp_pwd($handle);
echo "Name des aktuellen Verzeichnisses: " . $name;
```

Ausgabe:

```
Name des aktuellen Verzeichnisses: /home/kbrand
```

Die Funktion ftp_chdir()

Mit der Funktion `ftp_chdir()` ist es möglich, das aktuelle Verzeichnis auf dem angesprochenen Server zu wechseln. Das Zielverzeichnis wird entweder relativ zur aktuellen Position oder direkt vom Root-Verzeichnis aus als String übergeben.

```
$name = ftp_pwd($handle);
echo "Name des aktuellen Verzeichnisses: " . $name;
$bool = ftp_chdir($handle, "/home/kbrand/html");
if ($bool) {echo "<br>Das neue Verzeichnis: " . ftp_pwd($handle);}
```

Ausgabe:

```
Name des aktuellen Verzeichnisses: /home/kbrand
Das neue Verzeichnis: /home/kbrand/html
```

Die Funktion gibt bei Erfolg TRUE zurück, ansonsten den Wert FALSE. Die Rückgabe sollte immer überprüft werden, um unschöne Überraschungen zu vermeiden.

Die Funktion ftp_cdup()

Die Funktion `ftp_cdup()` wechselt vom aktuellen Verzeichnis eine Verzeichnisebene höher. Bei Erfolg gibt die Funktion TRUE zurück, ansonsten FALSE.

```
echo ftp_pwd($handle) ."<br>";
ftp_cdup($handle);
echo ftp_pwd($handle);
```

Ausgabe:

```
/home/kbrand
/home
```

Die Funktionen ftp_nlist() und ftp_rawlist()

Um Informationen über den Inhalt eines Verzeichnisses zu bekommen, müssen die Daten zuerst ausgelesen werden. PHP stellt zwei Funktionen zur Verfügung, die diese Aufgabe übernehmen und das Ergebnis in einem Array zur Verfügung stellen.

Die Funktion `ftp_nlist()` ist die einfachere der beiden Möglichkeiten: Sie liest die Datei- und Verzeichnisnamen der im Zielverzeichnis gespeicherten Daten aus und gibt sie ohne Kommentar als Array zurück.

```
$verzeichnis = "/home/kbrand/html";
echo "Verzeichnis: <b>". $verzeichnis ."</b><br><br>";
$vliste = ftp_nlist($handle, $verzeichnis);
foreach ($vliste as $var)
{
   echo $var . "<br>";
}
```

Ausgabe:

```
Verzeichnis: /home/kbrand/html
/home/kbrand/html/index.html
/home/kbrand/html/menu.html
/home/kbrand/html/webalizer
/home/kbrand/html/body.html
/home/kbrand/html/uns.html
/home/kbrand/html/anmeldung.html
/home/kbrand/html/news.html
/home/kbrand/html/webdesign.html
/home/kbrand/html/admin
/home/kbrand/html/img
/home/kbrand/html/chat
/home/kbrand/html/agb.html
/home/kbrand/html/php
```

Problematisch ist die Form der Rückgabe, da auf den ersten Blick nicht zu erkennen ist, ob es sich bei einem Element um ein Verzeichnis oder um eine Datei handelt. Eine einfache Möglichkeit festzustellen, welches Format das Ziel hat, ist der Versuch, das Element als neues Verzeichnis zu öffnen. Schlägt der Versuch fehl, handelt es sich um eine Datei oder einen Link.

```
$verzeichnis = "/home/kbrand/html";
echo "Verzeichnis: <b>". $verzeichnis .„</b><br><br>";
$vliste = ftp_nlist($handle, $verzeichnis);
foreach ($vliste as $var)
{
  echo $var;
  if(ftp_chdir($handle, $var)) {echo "
Verzeichnis)<br>";}
  else {echo " (Datei)<br>";}
}
```

Ausgabe:

```
Verzeichnis: /home/kbrand/html
/home/kbrand/html/index.html (Datei)
/home/kbrand/html/menu.html (Datei)
/home/kbrand/html/webalizer (Datei)
/home/kbrand/html/body.html (Datei)
/home/kbrand/html/uns.html (Datei)
/home/kbrand/html/anmeldung.html (Datei)
/home/kbrand/html/news.html (Datei)
/home/kbrand/html/webdesign.html (Datei)
/home/kbrand/html/admin (Verzeichnis)
/home/kbrand/html/img (Verzeichnis)
/home/kbrand/html/chat (Verzeichnis)
/home/kbrand/html/agb.html (Datei)
/home/kbrand/html/php (Verzeichnis)
```

Je nach Konfiguration der php.ini kann es zu Warnmeldungen kommen, wenn versucht wird ein nicht vorhandenes Verzeichnis zu öffnen. In diesem Fall ändern Sie die Fehlertoleranz des Interpreters (Kapitel 6: Fehlersuche) oder verwenden die folgende Funktion.

Der große Bruder der Funktion `ftp_nlist()` ist die Funktion `ftp_rawlist()`, die eine detaillierte Übersicht eines Verzeichnisses zurückgibt. Intern führt die Funktion das FTP-Kommando LIST aus, das eine Verzeichnisübersicht im UNIX-Stil erstellt. Das Rückgabearray der Funktion enthält praktisch alle Daten dieser Anfrage.

```
$vliste = ftp_rawlist($handle, $verzeichnis);
```

Jedes Arrayelement enthält folgende Angaben:

- Angabe über den Datentyp: d = Verzeichnis; - = Datei; l = Link
- Zugriffsrechte (r = lesen; w = schreiben; x = ausführen) der Benutzergruppen: User, Gruppe und Andere
- Benutzername des Besitzer
- Benutzergruppe
- Größe in Bytes
- Monat
- Tag
- Jahr (oder Uhrzeit, wenn es das aktuelle Jahr ist)
- Datei oder Verzeichnisname

Das folgende Beispiel zeigt die Anwendung der Funktion. Mit den HTML-Tags <PRE> wird sichergestellt, dass der Browser die Formatierung der Funktion übernimmt und die zusätzlichen Leerzeichen nicht ignoriert.

```
$verzeichnis = "/home/pressedienst/html";
echo "Verzeichnis: <b>". $verzeichnis;
echo "</b><br><br><pre>";
$vliste = ftp_rawlist($handle, $verzeichnis);
foreach ($vliste as $var)
{
   echo $var . "<br>";
}
echo "</pre>";
```

Zur Abwechselung wurde in diesem Programm ein anderes Verzeichnis auf dem gleichen Server ausgelesen. Die Daten im Array sind durch Leerzeichen voneinander getrennt und nehmen immer den gleichen maximalen Platz im Arrayelement ein. Durch geeignete Stringfunktionen können die Daten recht einfach voneinander getrennt werden.

Verzeichnis: **/home/pressedienst/html**

```
total 112
drwxr-xr-x   2 aburg    users        4096 Jan  4  2001 admin
-rw-r--r--   1 aburg    users       11792 Feb 19  2001 agb.html
-rw-r--r--   1 aburg    users        2328 Feb 19  2001 anmeldung.html
-rw-r--r--   1 aburg    users        2369 Feb 19  2001 anmeldunghilfe.html
-rw-r--r--   1 aburg    users        2748 Feb 19  2001 body.html
drwxrwxrwx   2 aburg    users        4096 Jan 24  2001 chat
-rw-r--r--   1 aburg    users        4527 Jan 12  2001 daten.html
-rw-r--r--   1 aburg    users        1462 Feb 19  2001 hilfe.html
drwxr-xr-x   8 aburg    users        4096 Feb 25  2001 img
drwxr-xr-x   2 aburg    users        4096 Oct 18 00:25 img2
drwxrwxrwx   2 aburg    users        4096 Feb 25  2001 imgthumb
-rw-r--r--   1 aburg    users        1683 Jan 12  2001 impressum.html
-rw-r--r--   1 aburg    users        1175 Jan 15  2001 index.html
-rw-r--r--   1 aburg    users         831 Jan 12  2001 index2.html
-rw-r--r--   1 aburg    users          10 Jan  9  2001 leer.html
-rw-r--r--   1 aburg    users        3132 Jan 12  2001 menu.html
-rw-r--r--   1 aburg    users         300 Jan  4  2001 news.html
drwxr-xr-x   4 aburg    users        4096 Nov  3 14:53 php
-rw-r--r--   1 aburg    users        1482 Feb 19  2001 qualitaet.html
-rw-r--r--   1 aburg    users        1904 Feb 19  2001 service.html
-rw-r--r--   1 aburg    users         711 Jan  9  2001 test.html
-rw-r--r--   1 aburg    users         295 Jan  2  2001 uns.html
-rw-r--r--   1 aburg    users        2152 Jan 12  2001 web2.html
drwxr-xr-x   2 aburg    users        4096 Nov  3 23:56 webalizer
-rw-r--r--   1 aburg    users        2030 Oct 18 00:24 webdesign.html
```

Abbildung 13.1: Die Funktion ftp_ rawlist() auf einem UNIX-Server

Die Funktion ftp_systype()

Die Funktion `ftp_systype()` erkennt den Betriebssystem-Typ auf dem FTP-Server und gibt ihn als String zurück.

```
$system = ftp_systype($handle);
echo $system;
```

Ausgabe unter Windows 2000:

```
Windows_NT
```

Ausgabe unter UNIX:

```
UNIX
```

Ausgabe unter Apple MacOS:

```
MacOS
```

Die Funktionen ftp_size() und ftp_mdtm()

Genauere Informationen über einzelne Dateien können mit den Funktionen `ftp_size()` und `ftp_mdtm()` abgefragt werden. Als Parameter wird der exakte Dateiname einer Datei übergeben, die im aktuellen Verzeichnis liegt. Existiert die Datei nicht, gibt die Funktion FALSE zurück.

```
$datei = "/home/pressedienst/html/index.html";
$time = ftp_mdtm($handle,$datei);
$size = ftp_size($handle,$datei);
echo  "Größe der Datei: $size Byte<br>";
echo "Datum der letzten Änderung: " . date("d.M Y H:i:s", $time);
```

Ausgabe:

```
Größe der Datei: 1191 Byte
Datum der letzten Änderung: 15.Jan 2001 19:05:20
```

Die Funktion `ftp_size()` gibt die Größe der Zieldatei in Byte zurück. Die Funktion `ftp_mdtime()` gibt das Datum der letzten Änderung als UNIX Timestamp zurück, der in einem weiteren Schritt in ein lesbares Format umgewandelt werden muss. Diese Funktion wird nicht von allen Servern unterstützt. Alternativ kann auch die Funktion `ftp_rawlist()` eingesetzt werden.

13.5 Datentransfer und Manipulation von Serverdaten

Der eigentliche Transfer der Dateien zwischen den Servern wird mit einer Handvoll Funktionen erledigt, die relativ leicht zu handhaben sind. Zusätzlich bietet das FTP-Protokoll die Möglichkeit, bestehende Daten auf dem Server zu verändern, neu anzulegen oder sogar zu löschen.

Für die folgenden Programmfragmente gilt dasselbe wie im letzten Kapitel: Es wird eine bestehende FTP-Verbindung vorausgesetzt, wie sie im ersten Kapitel besprochen wurde.

Die Funktionen ftp_mkdir() und ftp_rmdir()

Mit den Funktionen `ftp_mkdir()` und `ftp_rmdir()` ist es möglich, Verzeichnisse auf dem Server anzulegen oder auch wieder zu löschen. Voraussetzung für den Einsatz dieser Funktionen sind ausreichende Benutzerrechte auf dem entfernten Rechner, da in beiden Fällen ein Schreibzugriff auf dem Server vorgenommen wird.

```
$bool = mkdir($handle, "/home/kbrand/html/test");
if ($bool) {echo "Verzeichnis erstellt:<br>";}
echo ftp_chdir($handle, "/home/kbrand/html/test");
echo ftp_pwd($handle);
```

Ausgabe:

```
Verzeichnis erstellt:
/home/kbrand/html/test
```

Das Skript erstellt ein Verzeichnis mit dem Namen test auf dem entfernten FTP-Server. Wenn der Vorgang erfolgreich war, wird eine Bestätigungsmeldung ausgegeben und das neue Verzeichnis angezeigt.

Das Gegenstück zu `ftp_mkdir()` ist die Funktion `ftp_rmdir()`, die bestehende Verzeichnisse löscht. Voraussetzung für das Löschen ist, dass das Verzeichnis keine Dateien mehr enthält.

```
$bool = ftp_rmdir($handle, "/home/kbrand/html/test");
if ($bool) {echo "Verzeichnis wurde gelöscht!<br>";}
```

Ausgabe:

```
Verzeichnis wurde gelöscht!
```

Der Name des Verzeichnisses muss mit einem Schrägstrich beginnen, das heißt der Pfad muss vom Root-Verzeichnis des Servers angegeben werden, damit keine Missverständnisse auftreten können.

Die Funktionen ftp_get() und ftp_put()

Die Funktionen `ftp_get()` und `ftp_put()` dienen dem Datentransfer zwischen dem Server und dem Rechner, auf dem das PHP-Skript läuft. Mit `ftp_put()` wird eine Datei vom lokalen Rechner auf den FTP-Server kopiert. Die Dateinamen werden als Strings an die Funktion übergeben.

```
$bool = ftp_put($handle, $zieldatei, $lokalerName, $modus);
```

Für den Parameter `$modus` unterscheidet PHP zwei Konstanten, die festlegen, wie die Datei transferiert werden soll.

- `FTP_ASCII` (Übertragung als Textdatei)
- `FTP_BINARY` (Übertragung als Binärdatei)

Je nach Art der Datei muss der entsprechende Modus gewählt werden. Ist die Funktion ftp_put() erfolgreich, gibt sie TRUE zurück.

```
$datei = "data.txt";
$bool = ftp_put($handle, "/home/kbrand/html/$datei", $datei,
FTP_ASCII);
if($bool) {echo "Datei wurde erfolgreich hochgeladen!";}
```

Ausgabe:

```
Datei wurde erfolgreich hochgeladen!
```

Das Skript lädt die Datei data.txt aus dem aktuellen Verzeichnis in das HTML-Verzeichnis des FTP-Servers. Als Übertragungsmodus wurde `FTP_ASCII` gewählt, da es sich dabei um eine Textdatei handelt.

Die Funktion `ftp_get()` ist das genaue Gegenstück zu `ftp_put()`, denn sie lädt eine Datei vom FTP-Server auf den lokalen Rechner. Das Vorgehen ist allerdings genauso wie bei der vorhergehenden Funktion. Der einzige Unterschied liegt in der Reihenfolge der Parameter: Bei `ftp_get()` wird zuerst die lokale Datei angegeben und dann der Pfad auf dem Server.

```
$datei = "data.txt";
$bool = ftp_get($handle, $datei, "/home/kbrand/html/$datei",
FTP_ASCII);
if($bool) {echo "Datei wurde heruntergeladen!";}
```

Ausgabe:

```
Die Datei wurde heruntergeladen
```

Das Skript lädt die Datei data.txt vom FTP-Server wieder auf den Rechner, wo das PHP-Skript läuft.

Die Funktion ftp_rename()

Die Funktion `ftp_rename()` benennt eine Datei im aktuellen Verzeichnis auf dem FTP-Server um. Neben dem FTP-Handle muss der Funktion der Name der Zieldatei übergeben werden sowie die neue Bezeichnung. Werden unterschiedliche Pfadangaben gemacht, wird die Datei nicht umbenannt, sondern mit neuem Namen in das neue Verzeichnis verschoben.

```
ftp_chdir($handle, "/home/kbrand/html");
$bool = ftp_rename($handle, "data.txt", "data.bak");
if ($bool) {echo "Datei wurde umbenannt!";}
```

Ausgabe:

```
Datei wurde umbenannt!
```

Ist die Funktion erfolgreich, gibt `ftp_rename()` den Wert TRUE zurück.

Die Funktion ftp_delete()

`ftp_delete()` löscht eine Datei im aktuellen Verzeichnis des FTP-Servers. Der Name der Zieldatei wird als String übergeben und darf auch Pfadangaben enthalten.

```
ftp_chdir($handle, "/home/kbrand/html");
$bool = ftp_delete($handle,"data.bak");
if($bool) {echo "Datei wurde gelöscht!";}
```

Ausgabe:

```
Datei wurde gelöscht!
```

Genau wie alle anderen FTP-Funktionen gibt `ftp_delete()` TRUE zurück, wenn sie erfolgreich war.

13.6 Übungen

1. Was bedeutet FTP und was unterscheidet dieses Protokoll vom HTTP-Protokoll?
2. Warum muss sich ein User für einen FTP-Transfer identifizieren?
3. Warum bietet FTP die Möglichkeit auf dem Server zu navigieren?
4. Was ist der Unterschied zwischen den Funktionen `ftp_chdir()` und `ftp_cdup()`?
5. Was ist der Unterschied zwischen den Funktionen `ftp_rawlist()` und `ftp_nlist()`?
6. Was ist bei der Funktion `ftp_rmdir()` zu beachten?

14 E-Mails mit PHP

Neben dem World Wide Web, das zu großen Teilen auf dem HTTP-Protokoll basiert, gehört der Datenaustausch per E-Mail zu den tragenden Pfeilern des Internets. Unterschiedliche Protokolle erlauben es, personalisierte Daten zwischen den verschiedenen Servern auszutauschen und Informationen gezielt bestimmten Personen zukommen zu lassen.

Im Gegensatz zu Webseiten, die im Normalfall die breite Masse der Internetnutzer ansprechen, gehört die E-Mail zu den präziseren Werkzeugen, um User einzeln kontaktieren zu können. Diese Möglichkeit macht den Einsatz von Mailingfunktionen in Skripten unerlässlich und sollte entsprechend genutzt werden.

Hinweis:

Für die Funktionen der IMAP-Bibliothek, die im Folgenden vorgestellt wird, ist es nötig, den PHP-Interpreter zu erweitern. Sie brauchen das kostenlose Modul `php_imap.dll`, das nicht in der Standarddistribution von PHP enthalten ist. Fragen Sie Ihren Systemadministrator, ob das IMAP-Modul auf Ihrem Server installiert ist oder ob er bereit ist, dieses Paket nachträglich zu installieren.

Möchten Sie das Modul lokal nutzen, können Sie es unter der folgenden Adresse kostenlos downloaden:

ftp://ftp.cac.washington.edu/imap/windows/

Kopieren Sie die Datei *php_imap.dll* in das Systemverzeichnis Ihres Windowsbetriebssystems und aktualisieren Sie die Konfigurationsdatei php.ini. Die Datei muss die folgenden zwei Einträge beinhalten:

```
extension_dir=c:\windows\system
extension=php_imap.dll
```

Das Verzeichnis, in dem der PHP-Interpreter nach den Erweiterungsmodulen suchen soll, muss an den Systempfad Ihres Rechners angepasst werden. Unter Windows NT oder Windows 2000 heißt der Systempfad standardmäßig

```
c:\winnt\system32
```

Nachdem Sie alle nötigen Einstellungen vorgenommen haben, muss der Apache Server neu gestartet werden. Danach sollten Ihnen alle Features des neuen Moduls zur Verfügung stehen.

14.1 Kapitelübersicht

- Verschiedene Protokolle
- Die Funktion `mail()`
- Das IMAP-Modul

14.2 Verschiedene Protokolle

Technisch gesehen muss zwischen verschiedenen Protokollen zum Austausch von E-Mail-Daten unterschieden werden. Im folgenden Kapitel werde ich kurz einige der wichtigsten Nachrichtenprotokolle vorstellen, die zur Zeit im Internet verwendet werden.

14.2.1 SMTP

Das *Send Mail Transfer Protocol* gehört zum Urgestein der Internettechnik und ist für das Senden von E-Mails zuständig. Immer wenn eine E-Mail geschrieben und verschickt wird, muss sie an einen SMTP-Server weitergeleitet werden. Dieser steht mit allen anderen Mailservern im Internet über das SMT-Protokoll in Kontakt, die sämtliche Nachrichten im Internet austauschen und so an den richtigen Empfänger weiterleiten.

Technisch gesehen passiert das über den TCP/IP Port 25, der speziell für diesen Zweck reserviert wurde. Das eigentliche Kernstück des Protokolls ist das SendMail-Programm, das auf jedem SMTP-Server verfügbar ist. Diese leistungsfähige Software hat die Aufgabe andere SMTP-Server zu kontaktieren und die Post der Internetuser weiterzuleiten, bis der Empfänger erreicht ist.

Ist die E-Mail angekommen, wird sie auf dem Zielserver gespeichert, bis der User sie über POP3, IMAP oder ein anderes Mailprotokoll abruft.

14.2.2 POP3

POP3 *(Post Office Protocol, Version 3)* ist wohl zur Zeit das verbreitetste Protokoll zum Transfer von E-Mail-Daten zwischen dem Mailserver und einem lokalen Rechner. Dieses Protokoll kommt immer dann zum Einsatz, wenn eine E-Mail per SMTP auf dem Mailserver eines Users eingetroffen ist. Um die Nachricht lokal verfügbar zu machen, wird sie über POP3 an den Rechner des Benutzers weitergeleitet und dort gespeichert.

Im Gegensatz zu SMTP, wo eine E-Mail ohne weitere Kontrolle von einem Server weitergegeben wird, ist es bei POP3 notwendig, sich gegenüber dem Server zu identifizieren. Dies ist vor allem dafür nötig, um die Nachrichten auf dem Server dem anfragenden User zuordnen zu können.

Neben dem Transport der E-Mails ist es über POP3 auch möglich, rudimentäre Verwaltungsaufgaben zu erledigen. Es können beispielsweise E-Mails auf dem Server gelöscht oder als Kopie dort gespeichert werden. Der Transfer der Daten geschieht über den TCP/IP Port 110.

14.2.3 IMAP

IMAP steht für *Internet Message Access Protocol* und ist der Quasi-Nachfolger von POP3. Im Gegensatz zum Vorgänger, bei dem die E-Mails nur auf dem Server zwischengespeichert wurden, kann man mit IMAP die E-Mails direkt auf dem Server bearbeiten. Damit ist der Zugriff von verschiedenen Arbeitsplätzen aus problemlos möglich, da es nicht zwingend notwendig ist, die E-Mails auf den lokalen Rechner herunterzuladen.

Außerdem können mehr oder weniger komplexe Ordnerstrukturen angelegt und verwaltet werden, so dass man nicht (wie bei POP3) auf den Posteingang beschränkt ist. Diese Eigenschaften machen IMAP wesentlich flexibler und leistungsfähiger als das POP3-Protokoll für sich genommen. Die Zukunft gehört sicher dieser Technik, auch wenn die Verwendung zur Zeit noch nicht so verbreitet ist.

Technisch gesehen arbeitet IMAP über den Port 143, der auch standardmäßig vom IMAP-Modul verwendet wird. Möchte man einen anderen Port verwenden, muss dieser explizit angegeben werden.

14.3 Die Funktion mail()

Die Funktion `mail()` dient zum Versenden von E-Mails über einen SMTP-Server. Die Funktion basiert auf dem SendMail-Programm, das bei jeder UNIX/Linux Distribution für Server mitinstalliert ist. Fehlt dieses Programm, wie zum Beispiel unter Windows, kann die Funktion nicht eingesetzt werden.

Der unschlagbare Vorteil von `mail()` ist die Unabhängigkeit von externen Modulen, wie zum Beispiel der IMAP-Bibliothek. Die Funktion ist standardmäßig in jedem PHP-Interpreter vorhanden.

```
$bool = mail($eMail, $titel, $text, $header);
```

Die Funktion ist sehr einfach zu handhaben, da sie mit typischen Elementen eines E-Mail-Clients arbeitet. Der erste Parameter übergibt die Zieladresse des Mailaccounts, an die die Nachricht geschickt werden soll. Typischerweise wird hier eine einfache E-Mail-Adresse verwendet, wie sie im Internet benutzt wird. Das Format orientiert sich an folgendem Aufbau:

```
name @ domain . topleveldomain (de, com , net, etc.)
```

Der zweite Parameter übergibt den Betreff der E-Mail als String, wie er beim Empfänger erscheinen soll. Der Inhalt der Mail sollte hier treffend zusammengefasst werden. Der dritte Parameter ist der eigentliche Inhalt der E-Mail, der verschickt werden soll. Diese Daten werden ebenfalls als String übergeben und können durch die bekannten Sonderzeichen `\n` und `\t` formatiert werden. Ein Zeilenumbruch wird mit \n realisiert und ein Tabulatorsprung über das Kürzel `\t`.

Im Prinzip wäre die E-Mail jetzt komplett, aber die Funktion `mail()` kann über einen optionalen Parameter erweitert werden. Der String `$header` erlaubt es, der Mail einige Charakterzüge zu verleihen, die im Normalfall von einem E-Mail-Client übernommen werden.

Header	Beschreibung
`From:`	Absender der E-Mail
`Reply-To:`	Antwortadresse
`X-Mailer:`	Mailapplikation
`X-Priority:`	Dringlichkeitsstufe

Tabelle 14.1: Weitere Header-Informationen für E-Mails

Alle diese Angaben sind optional und können auch weggelassen werden. Die Angaben, die im Header der E-Mail verwendet werden sollen, müssen zu einem einzigen String zusammengefasst werden, der durch Zeilenumbrüche `\n` getrennt ist. Das Schlüsselwort und die eigentliche Information werden durch einen Doppelpunkt zusammengefügt. Ein typischer Header könnte also so aussehen:

```
$header = "From:dirk@ammelburger.de\nReply-To:webmaster@ammelbur-
ger.de\nX-Mailer:PHPmail()\n";
```

Für die Festlegung der Priorität müssen folgende Angaben gemacht werden:

Priorität	Erklärung
1	hohe Priorität
3	mittlere Priorität
5	niedrige Priorität

Tabelle 14.2: Prioritäten einer E-Mail

Einige Mail-Clients unterstützen die Angaben nicht, sondern ignorieren sie einfach. Für die Programme Microsoft Outlook und Outlook Express muss folgende Headerinformation zusätzlich hinzugefügt werden, damit die Angaben zur Priorität richtig interpretiert werden.

```
MSMail-Priority: high, normal, low
```

Je nach Priorität muss eines der Stichwörter gewählt werden. Die Angabe muss mit der Priorität vom Schlüsselwort X-Priority übereinstimmen.

Das folgende Skript zeigt ein komplettes Beispiel zur Funktion `mail()`. Achten Sie darauf, dass die Funktion nur auf Systemen arbeitet, die das Programm SendMail verwenden. In der Regel ist dies nur auf UNIX- oder Linux-Rechnern der Fall.

```
<?
$eMail = "dirk@ammelburger.de";
$titel = "Dies ist eine Testmail!";
$text = "Sehr geehrte Damen und Herrn,\n wir wünschen Ihnen alles
Gute für den Rest Ihres Lebens.\n\nViele Grüße ...";
$header = "From:dirk@ammelburger.de\nReply-To:webmaster@ammelbur-
ger.de\nX-Mailer:PHPmail()\nX-Priority:1\nMSMail-Priority: high";
$bool = mail($eMail, $titel, $text, $header);
if ($bool) {echo "eMail wurde erfolgreich versandt!";}
?>
```

Listing 14.1: Eine E-Mail wird mit der Funktion mail() versandt

Ausgabe:

```
eMail wurde erfolgreich versandt!
```

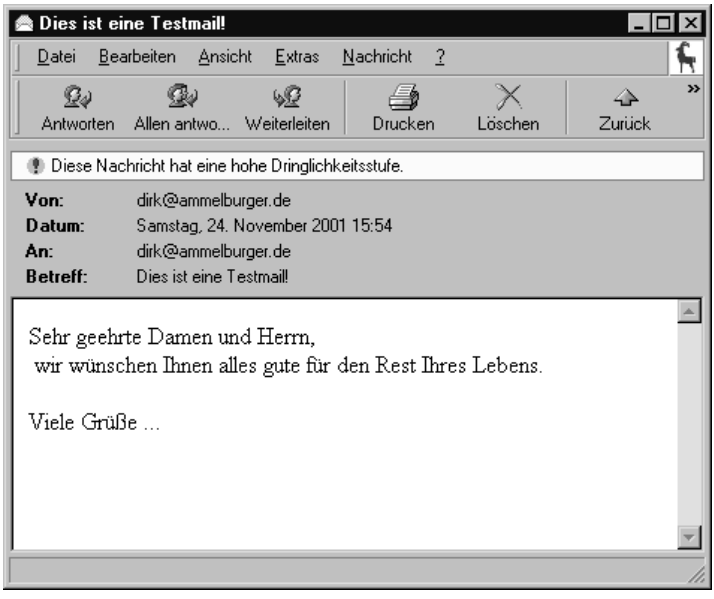


Abbildung 14.1: Die E-Mail ist angekommen

Die Funktion `mail()` ist eine wunderbare Sache, um unkompliziert aus einem Skript heraus eine E-Mail zu schreiben. Bestätigungsmails für Shopsysteme oder einfache Mailinglisten sind mit Sicherheit Anwendungsgebiete für diese Funktion. Auf der anderen Seite ist es mit den Hausmitteln von PHP nicht (oder nur sehr umständlich) möglich, E-Mails von einem Server abzurufen. Diese Lücke schließt das Modul IMAP.

14.4 Das IMAP-Modul

Das IMAP-Modul gehört zu den möglichen Zusatzbibliotheken, die von PHP unterstützt werden. Immer wenn Sie zusätzliche Möglichkeiten für den Austausch von E-Mails verwenden möchten, sollten Sie dieses Paket einbinden und die entsprechenden Treiber aktivieren. Eine Anleitung dazu finden Sie am Anfang dieses Kapitels.

Alle Funktionen des IMAP-Moduls beginnen mit dem Präfix *imap* und dienen zur Verarbeitung von Mailboxen, die auf einem IMAP-Server liegen. Aber diese Bibliothek kann weit mehr, denn sie unterstützt zusätzlich die Zusammenarbeit mit »normalen« POP3-Servern, die zur Zeit weit häufiger im Internet verwendet werden.

Das folgende Kapitel beschreibt den Einsatz der IMAP-Funktionen sowohl im Zusammenhang mit POP3-Mailboxen als auch für die vorgesehenen IMAP-Server. Da das IMAP-Protokoll weit mächtiger ist als POP3, werde ich nicht alle Funktionen vorstellen können und stellenweise Schwerpunkte setzen müssen. Das Ziel des Kapitels ist es, mit dem IMAP-Modul einen rudimentären Mail-Client für das Web zu programmieren, der es erlaubt, E-Mails von POP3-Mailboxen abzurufen.

14.4.1 Kontakt mit einer Mailbox

Der erste Schritt zum Datenaustausch zwischen den Servern ist der Aufbau einer Verbindung mit der entsprechenden Mailbox. Dieser Schritt wird über die Funktion `imap_open()` gesteuert, über die auch gleichzeitig die Art der Zielmailbox festgelegt wird.

```
$handle = imap_open ($postfach, $user, $passwd, $flags);
```

Die Funktion gibt ein Verbindungs-Handle oder FALSE bei eventuell auftretenden Fehlern zurück. Verbindungen können nicht nur zu IMAP-, sondern auch zu POP3-Servern aufgebaut werden, wobei dann allerdings nicht der gesamte Funktionsumfang zur Verfügung steht. Je nach Art der kontaktierten Mailbox muss der Verbindungsstring für den Wert `$postfach` angepasst werden.

Der Name des Postfachs setzt sich zusammen aus der Bezeichnung des Servers sowie dem Pfad des Postfachs für diese Mailbox. Der spezielle Name INBOX steht dabei als Synonym für das persönliche Postfach des jeweiligen Benutzers. Die Bezeichnung des Servers setzt sich aus dem eigentlichen Servernamen oder der entsprechenden IP-Adresse sowie dem gewünschten Protokoll (getrennt durch '/') und optional den zu benutzenden Port (getrennt durch ':') zusammen. Die gesamte Serverbezeichnung wird in '{' und '}' eingeschlossen, gefolgt vom Pfadnamen.

Der Funktionsaufruf für den Kontakt zu einem IMAP-Server würde beispielsweise so aussehen:

```
$handle = imap_open("{imap.server.de:143}INBOX", "user", "passwd",
flags);
```

Derselbe Aufruf für eine POP3-Mailbox hat diese Form:

```
$handle = imap_open ("{mail.server.de/pop3:110}INBOX", "user",
"passwd");
```

Die Benutzerdaten für die Werte User und Passwort sind dieselben Angaben, die auch in den bekannten Mail-Clients gemacht werden müssen, um E-Mails von einem Server abrufen zu können. Diese Daten werden Ihnen in der Regel mitgeteilt, wenn das neue Postfach angelegt wird.

Der optionale Parameter *flags* kann sich aus den folgenden Bitmasken zusammensetzen:

- `OP_READONLY` – über diese Verbindung darf nur lesend zugegriffen werden
- `OP_HALFOPEN` – es wird eine IMAP-Verbindung aufgebaut, aber noch kein Postfach ausgewählt
- `CL_EXPUNGE` – beim Schließen der Verbindung werden alle zum Löschen vorgemerkten Nachrichten endgültig gelöscht.

Das Gegenstück zu `imap_open()` ist die Funktion `imap_close()`, die eine bestehende Verbindung zu einem Mailserver wieder schließt.

```
imap_close($mbox, $flag);
```

Neben dem obligatorischem Handle auf die Zielverbindung kann auch hier ein optionaler Parameter übergeben werden. Der Wert `CL_EXPUNGE` sorgt dafür, dass die Mailbox vor dem Schließen geleert wird.

Mit diesen Funktionen ist ein erster Kontakt zu einer Mailbox möglich:

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user",
"passwd");
if ($mbox) {echo "Verbindung erfolgreich geöffnet!";}
imap_close($mbox);
?>
```

Listing 14.2: Kontakt zu einer POP3-Mailbox

Ausgabe:

```
Verbindung erfolgreich geöffnet!
```

Das Skript öffnet eine POP3-Mailbox auf dem Server mail.server.de über den Port 110 und schließt sie im nächsten Schritt wieder. Wenn eine Erfolgsmeldung ausgegeben wird, dann hat die Verbindung geklappt.

14.4.2 Informationen über eine Mailbox

Nachdem eine Verbindung erstellt wurde und die Funktion `imap_open()` ein Mailboxhandle geschaffen hat, können die übrigen Funktionen des IMAP-Moduls genutzt werden. Der erste Schritt nach dem Kontakt ist vermutlich festzustellen, ob E-Mails in der Mailbox gespeichert sind, und wenn ja, ob neue Nachrichten dabei sind.

Die Funktion `imap_check()` liefert einen Überblick über die kontaktierte Mailbox und gibt eine Reihe von nützlichen Informationen zurück. Die Daten der Funktion werden in einem Objekt gekapselt, das über die objektorientierte Syntax ausgelesen werden muss. Insgesamt sind es fünf Eigenschaften, die getrennt voneinander abgebildet sind.

```
$obj = imap_check($mbox);
```

Folgende Daten sind im Objekt `$obj` nach dem Funktionsaufruf gespeichert:

- `Date`: Datum und Zeit auf dem Server
- `Driver`: Mailprogramm auf dem Server
- `Mailbox`: Name der Mailbox
- `Nmsgs`: Anzahl der Nachrichten auf dem Server
- `recent`: Anzahl der neuen Nachrichten auf dem Server

Das folgende Skript zeigt den Einsatz der Funktion in der Praxis:

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user", "pwd");
$obj = imap_check($mbox);
echo "Serverzeit: ". $obj -> Date ."<br>";
echo "Anzahl der Nachrichten: ". $obj -> Nmsgs ."<br>";
echo "Programm: ". $obj -> Driver ."<br>";
echo "Name der Mailbox: ". $obj -> Mailbox ."<br>";
```

```
echo "Anzahl der neuen Nachrichten: ". $obj -> Recent;
imap_close($mbox);
?>
```

Listing 14.3: Informationen über eine Mailbox

```
Serverzeit: Sun, 25 Nov 2001 13:37:37 +0100 ((MEZ) Mitteleuropäische
Zeit)
Anzahl der Nachrichten: 5
Programm: pop3
Name der Mailbox: {mail.server.de:110/pop3/user="user"}INBOX
Anzahl der neuen Nachrichten: 5
```

Die Funktion ist sowohl für IMAP-Server als auch für POP3-Server einsetzbar. In der Anwendung gibt es keinen Unterschied. Die einzige Ausnahme ist die Angabe über die neuen E-Mails in einer Mailbox: Auf POP3-Servern kann nicht zwischen alten und neuen Mails unterschieden werden, darum wird jede Nachricht als neu eingestuft.

Zum Teil sind die Informationen über die Mailbox auch mit einzelnen Funktionen auszulesen. Die Anzahl aller Nachrichten kann beispielsweise mit der Funktion `imap_num_msg()` in Erfahrung gebracht werden.

```
$num = imap_num_msg($mbox);
```

Die Anzahl der Nachrichten wird als Integerwert zurückgegeben. Für IMAP-Server gibt es zusätzlich die Funktion `imap_num_recent()`, die die Anzahl der neuen Nachrichten zurückgibt.

```
$num = imap_num_recent($mbox);
```

Die Anzahl wird ebenfalls als Integerwert zur Verfügung gestellt. Die Funktion ist für POP3-Server einsetzbar. In diesem Fall wird allerdings derselbe Wert wie für die Funktion `imap_num_msg()` zurückgegeben.

Eine weitere Funktion, die ganz ohne objektorientierte Ansätze auskommt, ist `imap_mailboxmsginfo()`. Sie gibt ein Array mit allgemeinen Informationen über eine Mailbox zurück, die als einzelne Elemente ausgelesen werden können.

```
$liste = imap_mailboxmsginfo($mbox);
```

Das assoziative Array besteht aus acht Elementen, die sich wie folgt aufteilen:

- `Unread`: Ungelesene Nachrichten
- `Deleted`: Gelöschte Nachrichten

- `Nmsgs`: Anzahl der Nachrichten insgesamt
- `Size`: Größe aller Nachrichten in Byte
- `Date`: Serverzeit
- `Driver`: Mailprogramm
- `Mailbox`: Name der Mailbox
- `Recent`: Anzahl der neuen Nachrichten

Die Daten können recht einfach über die Schleife `foreach()` ausgelesen werden:

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "xxx", "123");
$liste = imap_mailboxmsginfo($mbox);
foreach ($liste as $ele => $var)
{
   echo "$ele: $var<br>";
}
imap_close($mbox);
?>
```

Listing 14.4: Informationen über eine Mailbox mit imap_mailboxmsginfo()

Ausgabe:

```
Unread: 5
Deleted: 0
Nmsgs: 5
Size: 8911
Date: Sun, 25 Nov 2001 14:06:09 +0100 ((MEZ) Mitteleuropäische Zeit)
Driver: pop3
Mailbox: {mail.server.de:110/pop3/user="postmaster%online-presse-
dienst.de"}INBOX
Recent: 5
```

Für die Anzahl der ungelesenen Nachrichten gilt dasselbe wie für die neuen Nachrichten in der Mailbox: Auf einem POP3-Server sind die Angaben immer gleich Gesamtanzahl der Nachrichten, da es keine Möglichkeit gibt diese zu markieren.

Eine sehr einfache Funktion ist `imap_ping()`, die eine geöffnete Mailbox »anpingt« und so überprüft, ob die Verbindung noch besteht.

```
$bool = imap_ping($mbox);
```

Die Funktion gibt TRUE oder FALSE zurück und zeigt so das Ergebnis der Überprüfung an.

```
$bool = imap_ping($mbox);
if ($bool) {echo "Verbindung besteht noch!";}
else {echo "Verbindung wurde unterbrochen!";}
```

Diese Funktion sollte dann eingesetzt werden, wenn die Verbindung zweifelhaft ist oder sich über einen längeren Zeitraum erstreckt. Fehler können so einfach abgefangen werden.

14.4.3 E-Mails empfangen

Der folgenden Abschnitt behandelt den eigentlich wichtigsten Teil im Datentransfer zwischen dem Mailserver und dem Rechner, auf dem das PHP-Skript läuft. Mit den Informationen aus dem letzten Kapitel können Sie jetzt zum wesentlichen Punkt in diesem Kapitel kommen: Wie werden die Daten einer E-Mail ausgelesen?

Im letzten Kapitel haben Sie verschiedene Möglichkeiten kennen gelernt, wie Sie die Anzahl von E-Mails in einer Mailbox feststellen können. Diese Information ist relativ wichtig, weil sie im direkten Zusammenhang mit den E-Mail-Nummern steht, über die die einzelnen Nachrichten identifiziert werden.

Das Prinzip ist sehr einfach: Die erste Nachricht, also die Nachricht, die schon am längsten in der Mailbox gespeichert ist, hat die Nummer 1. Alle folgenden Mails werden in chronologischer Reihenfolge sortiert und entsprechend durchnummeriert. Die letzte E-Mail hat also die Nummer, die gleich der Anzahl der E-Mails in der Mailbox ist. Wird eine E-Mail gelöscht, dann wird beim nächsten Kontakt mit der Mailbox die Lücke automatisch geschlossen und die Nummern werden neu verteilt.

Wie die Nummern im Detail vergeben werden, spielt in der Praxis eigentlich kaum eine Rolle. Das Einzige, was Sie wissen müssen, ist die Nummer der letzten E-Mail, damit Sie nicht über das Ziel schießen. Über die speziellen Nummern ist es jetzt möglich, die einzelnen E-Mails herauszupicken und mit geeigneten Funktionen zu bearbeiten.

Funktion	Beschreibung
`imap_header()`	liest den Kopf einer Nachricht aus
`imap_headers()`	liest den Kopf aller Nachrichten aus und gibt die Daten als Array zurück
`imap_body()`	liest den Inhalt einer E-Mail aus

Tabelle 14.3: Informationen über E-Mails

Die Funktion `imap_header()` ist eine recht einfache Funktion, die im Gegenzug ein sehr komplexes Rückgabeobjekt generiert. Mit der Funktion ist es möglich, alle wichtigen und auch eine Reihe von unwichtigen Informationen aus dem Header einer Mail auszulesen.

```
$obj = imap_header($mbox, $num);
```

In der einfachsten Version brauchen lediglich das Mailboxhandle sowie die Nummer der betroffenen E-Mail übergeben zu werden. Die Funktion gibt das Ergebnis als ein Objekt mit einer Reihe von Eigenschaften zurück. Zusätzlich unterstützt die Funktion zwei optionale Parameter, die es erlauben, Teile der Rückgabe zu formatieren.

```
$obj = imap_header($mbox, $num, $froml, $subjectl);
```

Über den Parameter `$froml` kann der Absender der E-Mail auf eine bestimmte Anzahl von Zeichen reduziert werden. Ist dieser Parameter gesetzt, hat das Rückgabeobjekt eine weitere Eigenschaft mit dem Namen *fetchfrom*. Dasselbe gilt für den zweiten Parameter `$subjectl`, über den die Länge des Betreffs festgelegt werden kann. Dieser Parameter generiert die Objekteigenschaft *fetchsubject*. Beide Parameter werden als Integerwert übergeben.

Das Rückgabeobjekt besteht aus einer ganzen Reihe von Eigenschaften, von denen ich die wichtigsten hier vorstellen werde. Einige der Eigenschaften sind keine spezifischen Werte, sondern nummerische Arrays, die wiederum weitere Objekte mit entsprechenden Eigenschaften enthalten.

1. `date`: Datum und Zeit der E-Mail-Ankunft
2. `subject`: Betreff der E-Mail
3. `toaddress`: E-Mail-Adresse des Empfängers
4. `message_id`: ID der Nachricht

5. `fetchfrom`: durch den Parameter $froml begrenzter Absender
6. `fetchsubject`: durch den Parameter $subjectl begrenzter Betreff
7. `to[]`: Array mit Objekten der »to«-Zeile:
   - `personal`: Name oder Bezeichner des Absenders
   - `mailbox`: Mailbox des Absenders (Zeichen vor dem @)
   - `host`: Host des Absenders (Zeichen nach dem @)
   - `fromaddress`: die Daten aus der »from«-Zeile
8. `from[]`: Array mit Objekten der »from«-Zeile:
   - `personal`
   - `mailbox`
   - `host`
   - `ccadress`: die Daten aus der »cc«-Zeile
9. `cc[]`: Array mit Objekten der »cc«-Zeile:
   - `personal`
   - `mailbox`
   - `host`
   - `bccaddress`: die Daten aus der »bcc«-Zeile
10. `bcc[]`: Array mit Objekten der »bcc«-Zeile
    - `personal`
    - `mailbox`
    - `host`
    - `replay_toaddress`: Daten aus der »replyTo«-Zeile
11. `reply_to[]`: Array mit Objekten der »replyTo«-Zeile
    - `personal`
    - `mailbox`
    - `host`
    - `senderaddress`: Daten aus der »sender«-Zeile

12. `sender[]`: Array mit Objekten der »sender«-Zeile

- `personal`
- `mailbox`
- `host`
- `return_path`: Daten aus der »returnPath«-Zeile

13. `return_path[]`: Array mit Objekten der »returnPath«-Zeile

- `personal`
- `mailbox`
- `host`
- `update`: UNIX-Timestamp des Sendezeitpunkts

Diese Masse von Angaben mag auf den ersten Blick sehr erschlagend wirken, ist aber durch die intelligente Strukturierung leicht zu handhaben. Bedenken Sie, dass Teile der Daten nicht immer definiert sein müssen, da viele Punkte optionale Angaben in einer E-Mail sind.

Wird beispielsweise das »cc«-Feld in einer E-Mail nicht ausgefüllt, dann ist das komplette Array `cc[ ]` nicht definiert. Werden hingegen zwei oder mehr Angaben gemacht, wird pro weiterem Empfänger ein Arrayelement mit Daten besetzt.

Das folgende Skript zeigt, wie die wichtigsten Header-Informationen einer E-Mail ausgelesen werden können:

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user", "pwd");
$obj = imap_header($mbox, 2);
echo "Datum: ". $obj -> date ."<br>";
echo "Betreff: ". $obj -> subject ."<br>";
echo "Empfänger: ". $obj -> toaddress ."<br>";
echo "Mail-ID: ". $obj -> message_id ."<br>";
echo "Absender: ". $obj -> from[0]->personal ."<br>";
echo "Absender eMail: ". $obj -> from[0]->mailbox ."@";
echo $obj -> from[0]->host ."<br>";
imap_close($mbox);
?>
```

Listing 14.5: Header-Informationen einer E-Mail

Ausgabe:

```
Datum: Sun, 25 Nov 2001 13:04:19 +0100
Betreff: morgen sehen wir uns...
Empfänger: postmaster@online-pressedienst.de
Mail-ID: <001501c175a9$79dc7b60$0c64a8c0@master>
Absender: Gaby H.
Absender eMail: lastcode@freenet.de
```

Wie Sie im Beispiel sehen können, ist der Zugriff auf die Objekte in den einzelnen Arrays recht pragmatisch gelöst worden.

```
$obj -> from[0]-> mailbox
```

Der lockere Umgang von PHP mit typisierten Daten macht es möglich, dass Objekteigenschaften und Arrays ohne Probleme kombiniert werden können. Der Zugriff auf die Daten erfolgt einfach über die strukturierte Zuordnung der einzelnen Elemente, wie sie im oberen Diagramm dargestellt wurden.

Die Funktion imap_headers()

Eine wesentlich einfachere Methode, die Header-Daten von E-Mails in einem Postfach auszulesen, ist die Funktion `imap_headers()`. Diese Funktion liest die Informationen aller Mails auf dem Server auf einmal aus und stellt die Daten in einem Array zur Verfügung.

```
$mails = imap_headers($mbox);
```

Der Preis für diese einfache Handhabung sind die sehr schlicht formatierten Daten der Rückgabe, die bei weitem nicht den Informationsgehalt der Funktion `imap_header()` haben. In machen Situationen kann die Funktion aber trotz allem sehr nützlich sein.

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user", "pwd");
$mails = imap_headers($mbox);
foreach ($mails as $var)
{
   echo $var ."<br>";
}
imap_close($mbox);
?>
```

Listing 14.6: Die Funktion imap_headers()

Ausgabe:

```
N 1)22-Nov-2001 Dirk Ammelburger test (2516 chars)
N 2)23-Nov-2001 Gaby H. morgen sehen wir uns... (2120 chars)
N 3)25-Nov-2001 Dirk Ammelburger test2 (1220 chars)
N 4)25-Nov-2001 Verlag PHP4(1371 chars)
```

Jedes Arrayelement enthält die zusammengefassten Header-Daten einer E-Mail. Die Angaben setzen sich zusammen aus der Nummer der Mail gefolgt vom Datum und vom Namen des Absenders. Die beiden letzten Punkte sind der Betreff der Mail sowie eine Größenangabe in Byte. Die Formatierung der Daten wird automatisch vorgenommen, so dass man keinen Einfluss auf die Ausgabe hat.

Die Funktion imap_body()

Die Funktion `imap_body()` liest den Nachrichtentext einer Mail aus und gibt ihn als String zurück. Genau wie bei der Funktion `imap_header()` wird die Nachricht anhand der Nummer in der Mailbox ausgewählt.

```
$text = imap_body($mbox, $num);
```

Der zurückgegebene String entspricht den Daten, wie der Absender sie in das Textfeld eines Mail-Clients eingegeben hat.

```
<?
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user", "pwd");
$text = imap_body($mbox, 1);
echo nl2br($text);
imap_close($mbox);
?>
```

Listing 14.7: Die Funktion imap_body()

Ausgabe:

```
Dies ist eine Testmail!
Dies ist die zweite Zeile der Testmail...
```

Im Gegensatz zu den vorhergehenden Programmen wirkt dieses kleine Skript fast zu einfach um zu funktionieren. Doch auf den ersten Blick scheint alles in Ordnung zu sein. Selbst die Zeilenumbrüche werden dank der Funktion `nl2br()` korrekt wiedergegeben.

Trotzdem gibt es ein Problem: Solange die E-Mails als reiner Text in der Mailbox ankommen, werden Sie keine Probleme mit diesem Skript haben. Schwierig wird es allerdings, wenn Sie speziell formatierte E-Mails empfangen, die bestimmte Anforderungen an das Mailprogramm stellen.

14.4.4 E-Mail-Formate

Der heutige Standard für den Austausch von Daten per E-Mail basiert auf der Definition RFC 822 (Request for Comments), die das genaue Format einer elektronischen Mail festlegt. In der ursprünglichen Form darf eine E-Mail nur 7Bit-ASCII-Daten enthalten. Das hat zur Folge, dass keine Sonderzeichen verwendbar sind, wie zum Beispiel Umlaute oder andere deutsche Sonderzeichen.

Diesem Standard wurden verschiedene Erweiterungen hinzugefügt wie beispielsweise die Übertragung von Sonderzeichen oder die so genannten Multimedia Mails. Diese Erweiterungen wurden so gestaltet, dass auch ein Mailservers der nur 7Bit-ASCII-Daten unterstützt, diese Mails ohne Probleme weiterleiten kann. Dies wird durch den so genannten MIME-Standard ermöglicht.

Der MIME-Standard (RFC 1521) definiert einige zuzätzliche Zeilen im Header einer E-Mail. Diese Header-Zeilen werden von einem Mail-Client entsprechend interpretiert und dargestellt. Sender und Empfänger müssen natürlich über eine Mail-Software mit MIME-Unterstützung verfügen, um dieses Feature nutzen zu können.

Die folgenden Header-Zeilen werden im RFC 1521 definiert:

- `Mime-Version`
- `Content-Type`
- `Content-Transfer-Encoding`
- `Content-ID`
- `Content-Description`
- `Content-Disposition`

Der Content-Type-Header gibt an, um was für Daten es sich handelt. Für den Content-Type gibt es mehrere Type und Untertypen, die entsprechend kombiniert werden können. Hier eine kleine Liste der wichtigsten Content-Typen:

Content-Type	Beschreibung
text	Textnachricht
multipart	Die Mail besteht aus verschiedenen Teilen.
message	Newsserver-Nachricht
application	Die Mail enthält Daten, die von einem Programm ausgeführt werden müssen.
image	Die Mail enthält ein Bild.
audio	Die Mail enthält Sounddaten.
video	Die Mail enthält Videodaten.

Tabelle 14.4: Content-Type von E-Mails

Um die zusätzlichen Daten in einer E-Mail verlustfrei transportieren zu können, müssen sie durch bestimmte Formate codiert werden. Die Art der Codierung wird durch die Header-Zeile Content-Transfer-Encoding festgelegt. Insgesamt sind fünf Codierungsformate definiert:

Codierung	Beschreibung
7Bit	einfache Textmail
quoted-printable	einfacher Text, aber Sonderzeichen werden codiert.
base64	Daten sind mit dem base64-Algorithmus codiert.
8Bit	einfache Textmail, aber 8Bit codiert.
binary	Binäre Daten

Tabelle 14.5: Codierung von E-Mail-Daten

Je nach Inhalt der E-Mail wird eine dieser Codierungsmöglichkeiten genutzt, damit die Daten fehlerfrei über das Internet transportiert werden können. Einfache Textmails, die beispielweise Sonderzeichen enthalten, werden entweder als »quoted-printable« oder als »8Bit«-Codierung versandt. Bilder dagegen müssen im »base64«-Format codiert werden, weil es sich um binäre Daten handelt.

Der Trick bei all diesen Codierungen besteht darin, dass das Ergebnis ausschließlich aus ASCII-Zeichen besteht, die von jedem Mailserver im Internet verstanden werden. Alle Daten werden sozusagen auf das 7Bit-

Niveau einer einfachen Textmail reduziert. Der Mail-Client des Empfängers hat im Gegenzug dann die Aufgabe die codierten Daten wieder zu entschlüsseln, um sie korrekt darstellen zu können.

Was bedeuten diese Informationen nun für einen PHP-Mail-Client? In Prinzip nichts anderes als für jeden anderen Mail-Client auch: Jede E-Mail, die vom Programm angezeigt werden soll, muss zuerst auf ihr Format hin überprüft werden. In einem zweiten Schritt müssen die Daten decodiert werden, bevor sie letztendlich auf dem Bildschirm landen.

Die Funktion imap_fetchstructure()

Um den MIME-Type und die Codierung einer E-Mail feststellen zu können, muss der Header der Nachricht analysiert werden. Das IMAP-Modul hat eine eigene Funktion im Angebot, die diese Aufgabe für den Programmierer recht einfach macht. Ähnlich wie bei der Funktion `imap_header()` besteht die Rückgabe der Funktion aus einem komplexen Objekt, das alle Daten in Arrays und Eigenschaften kapselt.

```
$obj = imap_fetchstructure($mbox, $num);
```

Neben dem Handle auf eine geöffnete Mailbox übernimmt die Funktion nur noch die Nummer der E-Mail, die analysiert werden soll. Das Rückgabeobjekt besteht aus folgenden Eigenschaften:

- `type`: primärer MIME-Type der Nachricht
- `encoding`: Transfercodierung
- `ifsubtype`: TRUE, wenn das Feld subtype definiert ist
- `subtype`: MIME-Subtype
- `lines`: Anzahl der Zeilen des Mailteils
- `size`: Größe in Bytes dieses Mailteils
- `ifdescription`: TRUE, wenn description-String definiert ist
- `description`: Beschreibungsstring
- `disposition`: Position in der E-Mail
- `ifid`: TRUE, wenn eine ID definiert ist
- `id`: ID
- `ifparameters`: TRUE, wenn Parameter definiert sind

- `parameters`: Array mit MIME-Parametern
- `parts`: Array mit Objekten, die die einzelnen Teile der Mail beschreiben

Je nach Mailtyp kann es passieren, dass bestimmte Eigenschaften des Rückgabearrays nicht definiert sind. Die Informationen sind für diese Mail dann nicht notwendig, und sie kann auch ohne diese korrekt interpretiert werden.

Über das Attribut *type* des Rückgabeobjekts kann der primäre MIME-Type der E-Mail festgestellt werden. PHP codiert die verschiedenen Typen über einen Integerwert zwischen 0 und 7, die wie folgt zu interpretieren sind:

Nummer	MIME-Type
0	text
1	multipart
2	message
3	application
4	audio
5	image
6	video
7	other

Tabelle 14.6: MIME-Types in PHP

Gibt die Funktion den MIME-Type 1 zurück, also »multipart«, dann wird automatisch das Objektarray `parts[]` definiert, in dem die einzelnen Teile der Nachricht gespeichert sind. Jedes Objekt im Array `parts[]` muss komplett eigenständig behandelt werden und verfügt über dieselben Eigenschaften wie das Hauptobjekt der Mail.

Das Attribut »encoding« eines Mailobjekts gibt die jeweilige Transfercodierung dieses Teils der E-Mail zurück. Auch hier verwendet das IMAP-Modul Integerzahlen, die die Angaben abkürzen.

Nummer	Codierung
0	7Bit
1	8Bit
2	Binary
3	Base64
4	quoted-printable
5	other

Tabelle 14.7: Transfercodierung in PHP

Je nach Codierung muss dieser Teil der E-Mail durch eine bestimmte IMAP-Funktion decodiert werden, damit er korrekt verarbeitet werden kann.

Das assoziative Array *parameters* besteht aus einer Reihe von Objekten, die die Attribute »attribute« und »value« besitzen. Jedes Objekt steht für einen MIME-Type einer MULTIPART E-Mail und speichert eventuelle Parameter.

Eine weitere wichtige Information ist die Objekteigenschaft *disposition*, über die ein Mailteil identifiziert werden kann. Eine MIME-Mail kann sich in verschiedene Bereiche, wie zum Beispiel den Text und das Attachment, aufteilen, die später gesondert decodiert werden müssen.

Nach so viel Theorie wird es Zeit, mit einem Beispiel die Anwendung dieser Techniken zu demonstrieren. Das folgende Skript hat die Aufgabe die Bestandteile einer E-Mail zu analysieren und für jeden Teil den MIME-Type sowie die Tranfsercodierung festzustellen.

```
<?
//Nummer der eMail, die analysiert werden soll!
$num = 7;
//Funktionsdefinitionen
function get_encoding($obj)
//Der Typ des Mailobjekts wird festgestellt und als String zurückge-
geben
{
   switch ($obj->encoding)
   {
      case 0: return "7Bit<br>";break;
      case 1: return "8Bit<br>";break;
```

```
      case 2: return "Binary<br>";break;
      case 3: return "Base64<br>";break;
      case 4: return "Quoted-printable<br>";break;
      case 5: return "other<br>";break;
      default: return "unbekannt";
   }
}
function get_mime($obj)
//Der MIME-Type des Mailobjekts wird festgestellt und zurückgegeben
{
   switch($obj->type)
   {
      case 0: return "text/". $obj->subtype ;break;
      case 3: return "application/". $obj->subtype ."<br>";break;
      case 4: return "audio/". $obj->subtype ;break;
      case 5: return "image/". $obj->subtype ;break;
      case 6: return "video/". $obj->subtype ;break;
      case 7: return "other/". $obj->subtype ;break;
      default: return "unbekannt";
   }
}
//Mailbox wird geöffnet
$mbox = imap_open ("{mail.agitos.de/pop3:110}INBOX", "user", „pwd");
//Die Mail wird analysiert!
$obj = imap_fetchstructure($mbox, $num);
//Der MIME-Type der Mail wird festgestellt.
if ($obj->type == 0)
{
   //MIME-Type text/html oder text/plain
   echo "MIME-Type: text/";
   echo $obj->subtype ."<br>";
   //Codierung wird festgestellt
   echo "Codierung: ";
   echo get_encoding($obj);
   //Da die Mail nicht multipart ist:
   echo "<br>Die Mail hat keine weiteren Bestandteile!";
}
if ($obj->type ==1)
{
   //MIME-Type multipart
   echo "Mail MIME-Type: multipart/";
   echo $obj->subtype ."<br><br>";
   $part=0;
   foreach($obj->parts as $var)
   {
      echo $part+1 .". Teil der Mail:<br>";
      echo "MIME-Type ". get_mime($var) . "<br>";
```

```
        echo "Codierung ". get_encoding($var);
        echo "<br><br>";
        $part++;
    }
}
//Mailbox wird geschlossen!
imap_close($mbox);
?>
```

Listing 14.8: Analyse einer E-Mail

Je nach E-Mail, die dem Skript zur Analyse übergeben wird, ändert sich die Ausgabe des Programms. Hier zwei Beispiele einer möglichen Ausgabe:

```
1. Teil der Mail:
MIME-Type text/PLAIN
Codierung Quoted-printable


2. Teil der Mail:
MIME-Type text/HTML
Codierung Quoted-printable
```

Bei dieser E-Mail handelt es sich um eine HTML-formatierte E-Mail, die aus einem alternativen Textpart besteht. Dieser kommt dann zum Einsatz, wenn der Mail-Client des Empfängers die HTML-Mail nicht interpretieren kann. Ansonsten wird dieser Part ignoriert.

```
Mail MIME-Type: multipart/MIXED

1. Teil der Mail:
MIME-Type TEXT/plain
Codierung 7Bit


2. Teil der Mail:
MIME-Type image/JPEG
Codierung Base64
```

Diese E-Mail besteht aus einem reinen Textteil, an den ein Bild im JPEG-Format angehängt wurde. Da beide Teile der E-Mail interpretiert werden müssen, ist der Subtype als MIXED definiert. Im Gegensatz zur ersten Mail hat der Mail-Client hier nicht die Wahl, welchen Teil er darstellen soll.

Das Skript arbeitet im Prinzip mit allen Techniken, die im letzten Kapitel besprochen wurden. Die wichtigsten Teile wurden in zwei Funktionen zusammengefasst, die den MIME-Type und die Transfercodierung der Mails feststellen. Sowohl die Funktion `get_mime()` als auch die Funktion `get_encoding()` bekommen als Parameter ein Mailobjekt übergeben, dass sie analysieren sollen. Aus welchem Teil der Mail das Objekt stammt, spielt für die Funktion keine Rolle, da jeder Part der E-Mail die gleichen Eigenschaften hat.

Die Funktion imap_fetchbody()

Dank der Funktion `imap_fetchstructure()` wissen Sie nun, wie die Mail im Einzelnen aufgebaut ist. Um nun die einzelnen Teile der E-Mail korrekt decodieren zu können, brauchen Sie die Funktion `imap_fetchbody()`, die den Zugriff auf die Bestandteile der Mail gestattet.

```
$text = imap_fetchbody($mbox, $num, $part);
```

Der Aufbau der Funktion ist recht einfach und mit dem Vorwissen aus dem letzten Kapitel sehr einsichtig. Neben dem obligatorischen Mailboxhandle und der Nummer der Zielmail verlangt die Funktion als dritten Parameter einen String, der den zu analysierenden Bestandteil der Mail identifiziert.

Die Funktion orientiert sich dabei an der Struktur der Mail, wie sie durch die MIME-Header vorgegeben sind. Jeder Teil der Mail bekommt eine Nummer zugeordnet, über die man den jeweiligen Teil der Mails auslesen kann. Je nach Aufbau der E-Mail kann die Anzahl der Bestandteile schwanken, so dass man nicht immer von einem festen Grenzwert ausgehen kann.

Einige grundsätzliche Bestandteile sind allerdings jedes Mal gleich: Beispielsweise kann man davon ausgehen, dass der komplette Header immer über die Adresse »0« zu ermitteln ist.

```
$text = imap_fetchbody($mbox, $num, "0");
//Mail-header
```

Da neben dem Header noch mindestens ein Textteil existieren muss, ist die Adresse »1« ebenfalls immer gesetzt. Ab dem Punkt wird es allerdings schwierig, da der Inhalt von diesem Teil der Mail vom Gesamtaufbau abhängig ist. Besteht die Mail aus mehreren Teilen (multipart), kann es sein, dass hier ein Header zurückgegeben wird, der die untergeordneten Bestandteile dieses Teils der Mail beschreibt.

Um auf diese untergeordneten Teile zugreifen zu können, muss die Adresse genauer spezifiziert werden. Das geschieht über die Syntax der Punktnotation, die bereits aus der OOP bekannt ist.

```
$text = imap_fetchbody($mbox, $num, "1.1");
//untergeordneter Teil des Mailkörpers
```

Dieser Funktionsaufruf liest den ersten Teil des Mailkörpers einer E-Mail aus und speichert die Daten in der Variable $text.

Ein typisches Beispiel für diese Situation ist eine Mail mit Attachment. Der Aufbau gleicht einem kleinem Baumdiagramm, das hier die Verteilung der Mailbestandteile aufzeigt.

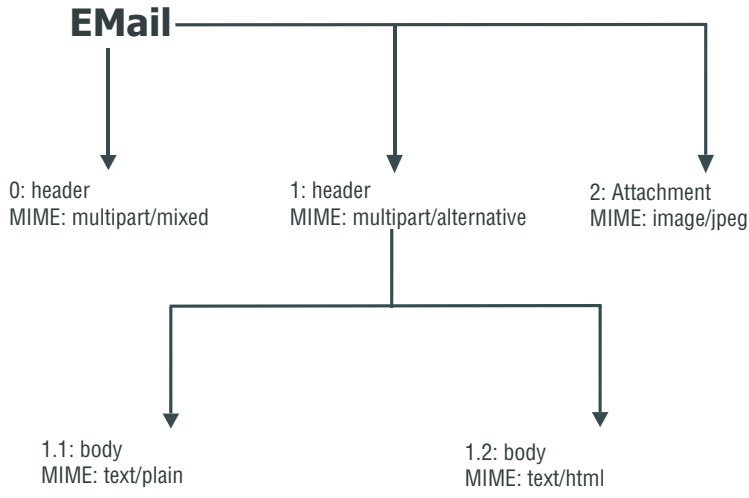


Abbildung 14.2: Aufbau einer multipart-E-Mail

Die Abbildung zeigt am Beispiel einer typischen E-Mail, die ein Bild im JPEG-Format angehängt hat, wie der Aufbau der verschiedenen MIME-Typen realisiert wird. Auf der ersten Ebene besteht die Mail aus zwei Komponenten, die im Header (bei 0) beschrieben werden. Diese Ebene wird mit dem MIME-Type multipart/mixed beschrieben, da unterschiedliche Bestandteile zur E-Mail gehören.

Typischweise wird das Attachment immer im letzten Teil der Mail gespeichert: In diesem Fall ist das Punkt 2, der unter dem MIME-Type image/

jpeg eine Grafik transportiert. Der interessanteste Punkt ist allerdings unter 1 zu finden, wo die Mail einen weiteren Header ausgibt. Im Unterschied zu ersten Header hat der Mail-Cient hier aber die Wahl, welchen Bestandteil der folgende Daten er verwenden kann (multipart/alternative).

Die Idee dahinter ist relativ einfach: Falls ein User eine E-Mail von einem Client aus verschickt, die HTML-Formatierungsanweisungen verwendet, besteht immer die Gefahr, dass der Empfänger diese nicht interpretieren kann. Für diesen Fall wird automatisch eine weitere Version der E-Mail im Format `text/plain` mitgeschickt. Der Empfänger hat nun die Wahl, welche Formatierung er anzeigen möchte, und kann im Zweifelsfall auf die neutrale Variante ausweichen.

Beide Versionen der E-Mail können über den Punkt 1 abgerufen werden. Die Version im MIME-Type text/plain hat die Adresse 1.1, und die HTML-codierte Variante kann über den Punkt 1.2 ausgelesen werden. Dazu ein kleines Beispiel:

```
<?
//Mailbox wird geöffnet
$mbox = imap_open ("{mail.mail.de/pop3:110}INBOX", "user", "pwd");
$num=8;
//header wird ausgelesen
$header=imap_fetchbody($mbox, $num, "0");
//zweiter header wird ausgelesen
$header2=imap_fetchbody($mbox, $num, "1");
//text/plain wird ausgelesen
$text=imap_fetchbody($mbox, $num, "1.1");
//text/html wird ausgelesen
$html=imap_fetchbody($mbox, $num, "1.2");
//$Bilddaten werden gespeichert
$bild=imap_fetchbody($mbox, $num, "2");
//Mailbox wird geschlossen!
imap_close($mbox);
?>
```

Listing 14.9: Bestandteile einer E-Mail werden ausgelesen

Das Skript liest die Daten der E-Mail aus und speichert sie in Variablen für die weitere Verwendung. Bevor die Daten allerdings ausgegeben werden können, müssen sie noch decodiert werden.

Decodieren von E-Mail-Daten

Am Anfang dieses Kapitels habe ich bereits erläutert, dass der Mailtransfer im Internet auf der Basis eines 7-Bit-Protokolls passiert, das weder Binär-

daten noch spezielle Sonderzeichen wie beispielsweise Umlaute unterstützt. Um diese Daten trotzdem verlustfrei übertragen zu können, wurden verschiedene Transfercodierungen entwickelt, die es erlauben eine Reihe von Formaten auf das 7-Bit-Niveau zu reduzieren, ohne dass dabei Daten verloren gehen.

Die Art der Codierung sowie der betroffene MIME-Type kann über den Header einer E-Mail ausgelesen werden. Mit diesen Informationen muss ein Mail-Client in der Lage sein, die codierten Formate wieder herzustellen. Das IMAP-Modul unterstützt für diese Zwecke eine Reihe von Decodierungsfunktionen, mit denen alle Transfercodierungen wieder rückgängig gemacht werden können.

Funktion	Beschreibung
`imap_base64()`	Decodiert eine Base64-codierten Text und gibt ihn als String zurück.
`imap_binary()`	Kovertiert einen 8Bit-Text in das Base64-Format.
`imap_qprint()`	Decodiert einen im Quoted-Printable-Format kodierten Text und gibt ihn als 8-Bit-String zurück.
`imap_8Bit()`	Konvertiert eine im 8-Bit-Format kodierte Zeichenkette in das Quoted-Printable-Format.
`imap_utf7_decode()`	Wandelt einen 7-Bit-String in einen 8-Bit-String um.
`imap_utf7_encode()`	Wandelt einen 8Bit String in einen 7-Bit-String um.

Tabelle 14.8: Codierung und Decodierung mit dem IMAP-Modul

Da PHP Ihnen den größten Teil der Arbeit abnimmt, ist der Umgang mit den codierten Bestandteilen einer E-Mail kein großes Problem. Je nach Transfercodierung muss die richtige Funktion verwendet werden, um die Daten in ein lesbares Format umzuwandeln.

Mit der Vorarbeit aus dem letzten Kapitel über die Funktion `imap_fetchbody()` sind Sie nun in der Lage die Beispielmail richtig formatiert auf dem Bildschirm auszugeben.

```
<?
//Mailbox wird geöffnet
$mbox = imap_open ("{mail.mail.de/pop3:110}INBOX",
“user", "pwd");
$num=8;
```

```
//text/html wird ausgelesen
$html=imap_fetchbody($mbox, $num, "1.2");
//$Bilddaten werden gespeichert
$data=imap_fetchbody($mbox, $num, "2");
//Ausgabe der decodierten Daten
echo "<b>TEXT(HTML):</b><br>";
echo imap_qprint($html);
echo "<br><br><b>ANHANG(Bild):</b><br>";
//Bilddaten werden decodiert
$data=imap_base64($data);
//Datei wird erstellt
$handle = fopen("data.jpg", "w");
//Bilddaten werden in die Datei geschrieben
fwrite($handle, $data);
//Datei wird geschlossen
fclose($handle);
//Ausgabe des Bildes
echo "<img src='data.jpg' border='1' width=300>";
//Mailbox wird geschlossen!
imap_close($mbox);
?>
```

Listing 14.10: Eine E-Mail wird mit Anhang ausgelesen

Ausgabe:

Abbildung 14.3: E-Mail mit Grafik

Das Skript konzentriert sich auf die beiden interessanten Teile der E-Mail, nämlich den HTML-codierten Text sowie das Bild im Anhang der Datei. Beide Datensätze werden im ersten Schritt durch die entsprechende Funktion decodiert und dann ausgegeben.

Während beim Text der E-Mail dieser Schritt recht einfach durch den Befehl `echo` umgesetzt wird, muss beim Anhang erst der Umweg über eine Datei gegangen werden. Über den Befehl `fwrite()` werden die Daten nach der Umwandlung in die geöffnete Datei »data.jpg« geschrieben. Im nächsten Schritt kann das Ergebnis dann einfach mit dem HTML-Tag <IMG...> im Browser präsentiert werden.

14.4.5 E-Mails auf dem Server löschen

Zu den Aufgaben eines Mail-Clients für POP3- oder IMAP-Server gehört das Löschen von alten oder unnötigen E-Mails im Postfach. Das IMAP-Modul stellt folgende Funktionen für diese Aufgabe zur Verfügung:

Funktion	Beschreibung
`imap_delete()`	markiert eine Nachricht zum Löschen in der Mailbox
`imap_expunge()`	löscht alle Nachrichten, die zum Löschen gekennzeichnet wurden
`imap_undelete()`	hebt die Kennzeichnung zum Löschen wieder auf

Tabelle 14.9: Löschfunktionen

Die Nachrichten in einer Mailbox können nicht sofort gelöscht werden, sondern müssen zuerst durch die Funktion `imap_delete()` gekennzeichnet werden. Die Zielmail wird durch den Parameter `$num` angegeben.

```
imap_delete($mbox, $num);
```

Durch `imap_delete()` wird die E-Mail für den eigentlichen Löschvorgang vorbereitet, der auf zwei verschiedenen Wegen möglich ist. Die erste Möglichkeit ist die Funktion `imap_expunge()`, die automatisch alle zum Löschen gekennzeichneten E-Mails einer Mailbox löscht.

```
imap_expunge($mbox);
```

Genau wie `imap_delete()` gibt die Funktion immer TRUE zurück, egal ob die Mailbox existert oder nicht. Ein weiterer Effekt von `imap_expunge()` ist die Neunummerierung der Mails in der Mailbox, um eventuell entstandene Lücken wieder zu schließen.

```
<?
//Mailbox wird geöffnet
$mbox = imap_open ("{mail.server.de/pop3:110}INBOX", "user", "pwd");
$num=3;
imap_delete($mbox, $num);
echo "Nachricht $num wurde markiert!<br>";
imap_expunge($mbox);
echo "Nachricht $num wurde gelöscht!";
//Mailbox wird geschlossen!
imap_close($mbox);
?>
```

Listing 14.11: Löschen von E-Mails

Ausgabe:

```
Nachricht 3 wurde markiert!
Nachricht 3 wurde gelöscht!
```

Die zweite Möglichkeit für das Löschen von markierten E-Mails kann mit der bereits besprochenen Funktion `imap_open()` automatisiert werden.

```
$mbox = imap_open ($mailbox, $user, $pwd, $flags);
```

Über den optionalen Parameter `$flags` kann das Bitfeld `CL_EXPUNGE` übergeben werden, das beim Schließen der Mailbox alle markierten E-Mails automatisch löscht.

Die Funktion `imap_undelete()` nimmt die Löschmarkierung einer E-Mail wieder zurück und verhindert damit, dass die Nachricht durch die Funktion `imap_expunge()` oder `imap_open()` gelöscht wird.

```
imap_undelete($mbox, $num);
```

Bereits gelöschte Mails können mit dieser Funktion nicht gerettet werden. Das Löschen einer Mail auf dem Server kann nicht wieder rückgängig gemacht werden.

14.5 Übungen

1. Was bedeutet SMTP und wo wird dieses Protokoll eingesetzt?
2. Wo liegt der Unterschied zwischen POP3 und IMAP?
3. Über welche Header-Bezeichnung wird die Dringlichkeitsstufe festgelegt?
4. Wo ist der Fehler im folgenden Programm?

```
<?
$mbox = imap_open ("{mail.server.de/pop3:143}INBOX", "user",
"passwd");
if ($mbox) {echo "Verbindung erfolgreich geöffnet!";
imap_close($mbox);
?>
```

5. Welche Informationen hat der Header einer E-Mail?
6. Was ist der Unterschied zwischen den Funktionen `imap_header()` und `imap_headers()`?
7. Welche Aufgabe hat der MIME-Type einer Mail?
8. Kann eine Mail mehrere Header haben?
9. Wie werden Bilder für den Mailversand codiert?
10. Warum werden Bilder für den Mailversand codiert?

15 XML

Auf den folgenden Seiten dieses Buches geht es um die Sprache XML und wie sie im Zusammenhang mit PHP sinnvoll eingesetzt werden kann. Jeder wird inzwischen etwas von dieser »Wundersprache« gehört haben, die durch die Medien geisterte, ohne wirklich Spuren zu hinterlassen. Was ist XML und wofür braucht man diese Sprache, wenn doch bisher auch ohne sie alles ganz gut funktioniert hat?

Diese und andere Fragen sollen beantwortet werden, bevor wir uns mit der eigentlichen Anwendung innerhalb von PHP beschäftigen werden. Natürlich ist es nicht möglich, innerhalb des folgenden Kapitels den kompletten Umfang und alle Möglichkeiten von XML zu besprechen, aber eine kurze Einführung wird Ihnen die wesentlichen Punkte des XML-Konzepts näher bringen.

15.1 Kapitelübersicht

- Die Story
- Was ist XML?
- Einführung in XML
- XML und PHP
- Das DOM-Modell

15.2 Die Story

Die Wurzeln von XML (*Extensible Markup Language*) reichen zurück bis in das Jahr 1986, in dem die Geburtsstunde der *Standard Generalized Markup Language* (SGML) liegt. Diese neue Sprache hat mit einem Schlag die Informationstechnologie aus den Fesseln der technischen Abhängigkeiten befreit und einen Weg eröffnet, Informationen ausschließlich auf der Basis ihres natürlichen Aufbaus zu vermitteln.

Wahre Größe konnte die Sprache durch das World Wide Web erlangen, das durch seinen heterogenen Aufbau das ideale Anwendungsgebiet von maschinenunabhängigem Informationstransfer ist. Die dabei verwendete Beschreibungssprache HTML (*Hypertext Markup Language*) ist ein direktes

Subset von SGML und demonstriert, wie die systemunabhängige Informationsmodellierung den weltweiten Datenaustausch erst möglich macht.

Aber auch in anderen Bereichen konnte SGML in derselben Zeit einige Erfolge verbuchen: In der *Text Encoding Initiative* (kurz: TEI) wurde eine SGML-Anwendung entwickelt, die zur Darstellung von literarischen, historischen und anderen geisteswissenschaftlichen Texten dient und erlaubt, Texte weltweit in einem einheitlichen Format auszutauschen und zu verwenden.

Gleichzeitig wurde mit dem Informationsmodell CALS (*Computer assisted lifecycle support*) eine Möglichkeit geschaffen wissenschaftliche Arbeiten und Periodika mitsamt Registern und Glossaren in SGML darzustellen. Inzwischen werden von technischen Handbüchern bis hin zu Produktdaten alle Informationen in diesem Format behandelt.

Etwa ein gutes Jahrzehnt später wurde 1998 mit XML (*Extensible Markup Language*) ein weiteres weit reichendes Subset von SGML geschaffen, das speziell für die Anforderungen in Netzwerken, besonders dem Internet, entwickelt wurde. Die Definition von XML basiert auf denselben Grundlagen wie SGML, ist jedoch viel knapper und logischer, da alle Merkmale, die in SGML kaum genutzt werden, einfach weggelassen wurden. Das Ziel, die Ausdrucksmöglichkeiten durch diese Beschneidung des Funktionsumfangs nicht einzuschränken, wurde so gut umgesetzt, dass XML inzwischen auch überall da eingesetzt wird, wo die Netzwerkfähigkeit nicht im Vordergrund steht.

XML ist im Prinzip nichts anderes als eine vereinfachte Version von SGML, die parallel genutzt werden kann. Jede XML-Codierung ist gleichzeitig auch eine SGML-Codierung und kann von jedem SGML-Parser interpretiert werden.

15.3 Was ist XML?

Die Extensible Markup Language (XML) ist also eine vereinfachte Version der Standard Generalized Markup Language (SGML), die bereits 1986 als Standard ISO 8879 verabschiedet wurde. Sowohl SGML als auch XML haben die Fähigkeit, Informationen strukturiert in einem systemunabhängigen Format zu modellieren und sind deshalb für den Datenaustausch in heterogenen Netzwerken, speziell dem Internet, geeignet.

Informationen können damit nicht nur in einem einheitlichen Format strukturiert werden, sondern die Strukturierungsprinzipien selbst sind auch durch ein formales Regelwerk (Grammatik) beschreibbar. Erst so werden weitergehende Verarbeitungsprozesse wie zum Beispiel Datenkonvertierung oder flexibles Navigieren in den Daten möglich.

Doch was heißt Informationsmodellierung? Grob gesagt geht es um die Strukturierung von Informationen anhand Gesetzmäßigkeiten, die wir in diesen Informationen finden. Im Gegensatz zu Datenbanken, die in ihrer Konzeption auf mehrdimensionalen Tabellen und Relationen basieren, richtet sich XML direkt nach der Struktur, die von den Informationen vorgegeben wird.

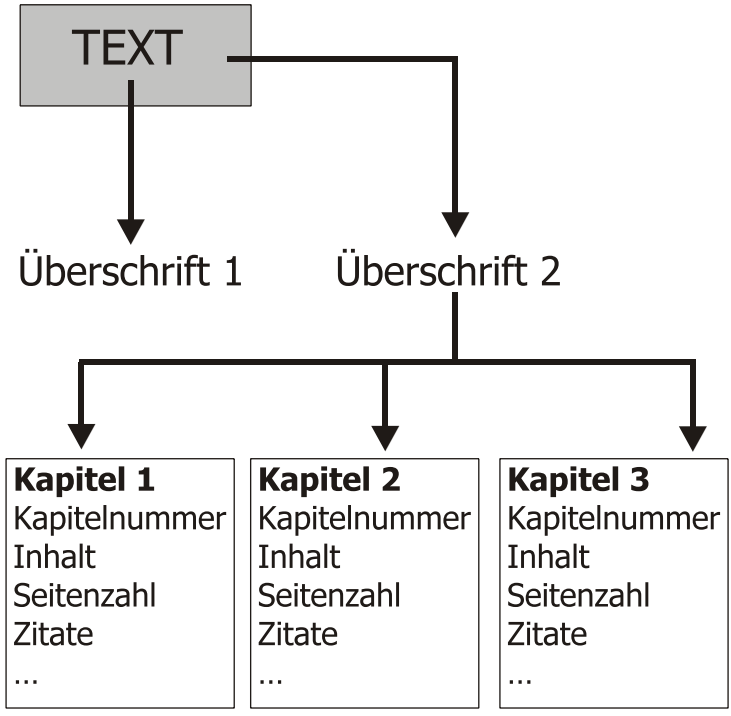


Abbildung 15.1: Informationsgliederung

Die Idee hinter dieser Art der Strukturierung geht auf verschiedene Beobachtungen zurück: Ein Text beispielsweise kann in unterschiedliche Ebenen aufgeteilt werden, wie Überschriften verschiedener Ordnungen oder übergeordnete Kapitel, die den Text gliedern. Gleichzeitig kann ein

Text semantisch in verschiedene Bereiche aufgeteilt werden, wie zum Beispiel Textinformationen, Zitate oder der Inhalt von Überschriften.

Das Besondere an dieser Strukturierung ist die Verteilung der Informationen, die in der Regel nicht flach aufgebaut sind, sondern wie ein Baumdiagramm einen tiefen Aufbau haben. Die verschiedenen Bereiche einer Informationen sind nicht zufällig verteilt, sondern gehorchen bestimmten »natürlichen« Regeln, die beispielsweise einen Text automatisch gliedern und aufteilen.

Das Beispiel zeigt den hierarchischen Aufbau eines Textes, der durch verschiedene Strukturen aufgegliedert ist. Alle Elemente dieses Aufbaus gehen von einem zentralen Wurzelelement aus, welches alle folgenden Informationseinheiten unter sich anordnet. Man unterscheidet dabei zwischen hierarchisch bezogenen Elementen, wobei eines der beiden Einheiten im selben Zweig höher steht als die andere Einheit, und zwischen sequenziellen Ordnungen, wo verschiedene Elemente auf derselben Ebene stehen.

Ordnung	Beispiel
sequenzielle Ordnung	Kapitel 1 bis 3, Überschriften
hierarchische Ordnung	Überschrift 2 zu jedem Kapitel

Tabelle 15.1: Ordnungen in in XML-Dokumenten

Der Aufbau dieses Beispiels ist typisch für die Darstellung von Informationen in ihrem natürlichen Auftreten. Datenbanken haben oft den Nachteil, dass sie Daten eine Struktur aufzwinken, die nicht immer optimal für eine weitere Verarbeitung geeignet sind. Dieses Problem besteht bei der Informationdarstellung durch XML nicht.

- Eine XML-Anwendung gibt an, was für Typen von Informationseinheiten es gibt und unter welchen Bezeichnern sie eindeutig identifiziert werden können.
- Informationen werden durch Regeln miteinander in Beziehung gesetzt, die in der Lage sind (tiefe) Baumstrukturen darzustellen.
- Die Gesamtheit der Regeln werden zu einer Grammatik zusammengefasst, die ebenfalls in XML dargestellt wird.

Ein XML-Dokument ist also in der Lage Informationen in jeder beliebigen Modellierung zu speichern. Die Grundlage für die Modellierung ist ein Regelwerk, das festlegt, welche Strukturen erlaubt sind und welche Zusammenhänge zwischen den Datensätzen existieren.

Weitere Informationen zu XML finden Sie unter *http://www.w3.org/XML* .

15.4 Einführung in XML

XML ist eine Metasprache, mit der andere Beschreibungssprachen definiert werden können. Das bedeutet, dass XML weder eigene Tags defniert noch eine bestimmte Grammatik hat, an die sich ein Autor halten muss, um mit XML arbeiten zu können. Aus diesem Grund ist XML beliebig erweiterbar und kann mit jedem Tag arbeiten, der nötig ist, um eine Aufgabe zu lösen.

Die Menge aller Tags in einer Sprache wird durch den Parser definiert, der Dokumente dieser Markupsprache interpretieren muss. Jeder Tag, den dieser Parser versteht, symbolisiert eine bestimmte Eigenschaft, die auf die Werte innerhalb dieser Tags übertragen werden.

Die Markupsprache HTML, das wohl bekannteste Subset von XML, definiert beispielsweise den Tag <B> für die Texteigenschaft BOLD (Fett) im Browser. Alle Daten, die innerhalb des öffnenden und schließenden Tags <B> stehen, werden automatisch mit der Eigenschaft BOLD versehen.

```
<B>Das ist ein Text in BOLD.</B>
```

Versucht man dasselbe aber mit einem Tag, der nicht in der HTML-Grammatik definiert ist, wird das nicht zum gewünschten Ergebnis führen.

```
<FETT>Das ist ein Text in FETT.</FETT>
```

Dieser Tag hat keine bestimmte Bedeutung. Die meisten Browser werden ihn zwar ignorieren, doch es können auch unerwartete Dinge passieren, wenn dieser Tag auf einen weniger flexiblen Parser stößt.

Dadurch, dass XML jeden Tag zulässt, ist sie beliebig erweiterbar (Extensible Markup Language: »Erweiterbare Auszeichnungssprache«). Das heißt, Sie können jeden Tag verwenden, der Ihnen gerade in den Sinn kommt oder den richtigen Namen für seine zukünftige Aufgabe hat. Die große Stärke von XML ist die Freiheit, jede beliebige Aussage durch beliebige Tags darstellen zu können, ohne durch bestehende Regeln eingeschränkt

zu werden. Ganz im Gegenteil: In XML definieren Sie Ihre eigenen Regeln für den jeweiligen Zweck.

Das folgende Beispiel zeigt ein gültiges XML-Dokument, das nur zum Zweck geschaffen wurde die grobe Gliederung eines Buches zu demonstrieren:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<Buch Titel="PHP und XML">
   <Ueberschrift num="1" text="Einführung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
         <Seitenzahl>68</Seitenzahl>
      </Kapitel>
      <Kapitel num="2">
         <Inhalt>Um ein Dokument zu parsen...</Inhalt>
         <Seitenzahl>120</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
   <Ueberschrift num="2" text="Anwendung">
      <Kapitel num="3">
         <Inhalt>...</Inhalt>
         <Seitenzahl>133</Seitenzahl>
      </Kapitel>
      <Kapitel num="4">
         <Inhalt>...</Inhalt>
         <Seitenzahl>89</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
</Buch>
```

Listing 15.1: Ein XML-Dokument

Wenn Sie noch nie ein XML-Dokument gesehen haben, dann mag Ihnen dieses Listing ein wenig merkwürdig vorkommen, da alle verwendeten Tags und Attribute frei erfunden sind. Es gibt keine Spezifikation, geschweige denn einen Parser, der ein Dokument mit diesen Tags interpretieren könnte, auch wenn es theoretisch möglich wäre.

Trotz allem handelt es sich hierbei um ein XML-Dokument. Solange Sie sich an einige Grundregeln halten, die von XML vorgeschrieben werden, können Sie die Struktur und den Inhalt der Daten beliebig bestimmen.

15.4.1 Tags in XML

Nach dieser langen Einführung wird es Zeit, dass wir uns den praktischen Teil von XML anschauen. Zu diesem Zweck werfen Sie bitte noch einmal einen Blick auf das Beispiel im letzten Kapitel. Auf den ersten Blick ähnelt die Syntax stark einem HTML-Dokument, auch wenn sämtliche Tags vollkommen aus der Luft gegriffen sind.

Alle Tags sind von spitzen Klammern umschlossen und unterscheiden sich so vom eigentlichen Inhalt des Dokuments. Gleichzeitig wird jeder geöffnete Tag (`<Kapitel>`) durch einen schließenden (`</Kapitel>`) wieder abgeschlossen. Im Gegensatz zu HTML ist XML in den Anforderungen an ein Dokument wesentlich strenger:

- Jeder Tag muss einen Abschluss-Tag besitzen. Der Abschluss-Tag muss in Inhalt und Form mit dem Start-Tag übereinstimmen. Es ist nicht möglich, den Tag `<Kapitel>` mit dem Abschluss-Tag `</kapitel>` zu schließen.
- XML unterscheidet zwischen Groß- und Kleinschreibung: In der Praxis bedeutet dies, dass der Tag `<GROSS>` sich vom Tag `<gross>` unterscheidet.

Abweichungen von diesen Regeln würde bei jedem Parser sofort eine Fehlermeldung provozieren.

Den Namen eines Tags dürfen Sie frei wählen, allerdings haben Sie auch hier einige Einschränkungen zu beachten. Ein Tag-Name darf nur aus

- Buchstaben
- Zahlen
- Unterstrichen

bestehen, wobei das erste Zeichen keine Zahl sein darf. Besonders Sonderzeichen wie zum Beispiel deutsche Umlaute oder Satzzeichen sollten in Tag-Namen vermieden werden, da XML sie nicht unterstützt.

```
Ä ä Ö ö Ü ü ß * + ! „ § $ % & / ( ) = ? / ~ # \ - . , ;: ^ ° @ < > |
```

Umlaute und das deutsche ß können in Tagnamen verwendet werden, wenn der Zeichensatz durch den ISO-Standard ISO-8859-1 erweitert wird. Dieses Vorgehen ist allerdings nicht empfehlenswert, da nicht jeder XML-Parser diese Codierung versteht und damit umzugehen weiß.

Attribute in Tags

Einige der Tags, die im Beispiel verwendet wurden, haben so genannte Attribute nach dem Tag-Namen, die weitere Informationen über den verwendeten Tag liefern.

```
<Ueberschrift num="1" text="Einführung">
```

Diese Technik wird mit Sicherheit aus HTML bereits bekannt sein, da die meisten Tags einer Webseite über einen oder mehrere Parameter verfügen. Ein bekanntes Beispiel ist der Link-Tag <A>, der über eine ganze Reihe Parameter verfügt.

```
<A HREF="link.html" TARGET="_blank">Klick</A>
```

Unter anderem wird innerhalb des Tags festgelegt, welches Ziel der Link haben soll und in welchem Browserfenster oder Frame das Ziel erscheint. Natürlich wäre es auch möglich gewesen, diese Informationen in Untertags an den Parser weiterzugeben, aber ein kurzer Blick auf das Beispiel zeigt, warum dieser Weg weit umständlicher ist.

```
<A>
   <HREF>link.html</HREF>
   <TARGET>_blank</TARGET>
   <TEXT>Klick</TEXT>
</A>
```

Vom erhöhten Schreibaufwand einmal abgesehen, der das Dokument für ein menschliches Auge schnell unübersichtlich macht, ist die erste Variante auch von der Struktur her vorzuziehen. Ohne die Unterteilung durch Attribute ist es nicht möglich festzustellen, welche Informationen nun zum <A>-Tag gehören und welche eigentlich nur weitere Informationen zu diesem Tag speichern.

Anhand von Attributen kann auch zwischen namensgleichen Tags unterschieden werden, ohne die weiterführende Baumstruktur analysieren zu müssen. Der Zugriff auf den Tag <Kapitel> mit dem Attributwert num = 1 ist wesentlich einfacher, als alle <Kapitel>-Tags zuerst zu analysieren, um den Untertag num = 1 zu finden. Attribute erlauben also eine bestehende

Baumstruktur mit zusätzlichen Informationen zu versehen, ohne den Aufbau zu verändern.

<?xml ?>-Tag

Jedes XML-Dokument beginnt mit dem `<?xml ?>`-Tag, dem eine besondere Bedeutung in jedem XML-Dokument zukommt. Die erste Auffälligkeit, die diesen Tag von den anderen Datentags unterscheidet, ist das Fragezeichen am Anfang und am Ende des Namens. Durch diese Kennzeichnung wird dieser bestimmte XML-Tag vom Parser als eine Verarbeitungsanweisung (*processing instruction: PI*) erkannt und speziell behandelt.

Je nach Dokument und Parser unterstützt XML eine ganze Reihe von PIs, von denen der <?xml ?>-Tag allerdings der wichtigste ist. Über ihn werden Informationen über die Version und den Codierungstyp der folgenden Daten angegeben.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Im obigen Beispiel wird die verwendete XML-Version mit dem Typ 1.0 festgelegt. Sollte ein Parser die Version nicht unterstützen, kann er anhand dieser Informationen die Verarbeitung des Dokuments ablehnen. Das zweite Attribut *encoding* legt den verwendeten Zeichensatz für das Dokument fest. Da in der vierten Zeile im Wort »Einführung« ein Umlaut verwendet wurde, muss der Zeichensatz um die deutschen Sonderzeichen erweitert werden. Ist das nicht der Fall, dann würde ein Parser eine Fehlermeldung auswerfen, da XML standardmäßig nur die ersten 128 Zeichen des ASCII-Zeichensatzes verwendet.

Eine PI hat die Aufgabe Informationen und Anweisungen an eine Applikation weiterzugeben, die mit den nachstehenden XML-Daten arbeiten soll. Zu diesem Zweck stehen die PI-Tags außerhalb des XML-Baums im so genannten Kopf des Dokumentes und werden durch das einleitende und abschließende Fragezeichen gekennzeichnet. PIs brauchen im Gegensatz zu Datentags keinen definierten Abschluss.

Alle PIs werden nach demselben Schema aufgebaut:

```
<?ziel anweisungen?>
```

Alle PIs, die xml als Ziel angegeben, gehören zur Standardmenge der von XML vorgegebenen Anweisungen. Jeder Parser sollte diese Angaben verarbeiten können. PIs können aber auch für bestimmte Informationen

selbst definiert werden, wenn XML-Dokumente nur von festgelegten Applikationen bearbeitet werden. Ist eine PI unbekannt, wird sie in der Regel vom Parser ignoriert.

15.4.2 XML-Code im Microsoft Explorer

Ein einfache und gleichzeitig effektive Möglichkeit, XML-Dokumente zu betrachten und auf mögliche Fehler zu kontrollieren, ist der Microsoft Explorer ab der Version 5. Je nach Betriebssystem und Konfiguration Ihres Rechners ist es möglich, dass XML-Dokumente (.xml) automatisch dem Explorer als Verarbeitungsapplikation zugeordnet werden.

Um ein XML-Dokument im Browser betrachten zu können, müssen Sie es wie ein HTML-Dokument auf den Browser ziehen (Drag&Drop) oder über die Befehlszeile vom lokalen Rechner aus aufrufen. Das oben besprochene Beispiel wird im Explorer 5.0 so dargestellt:

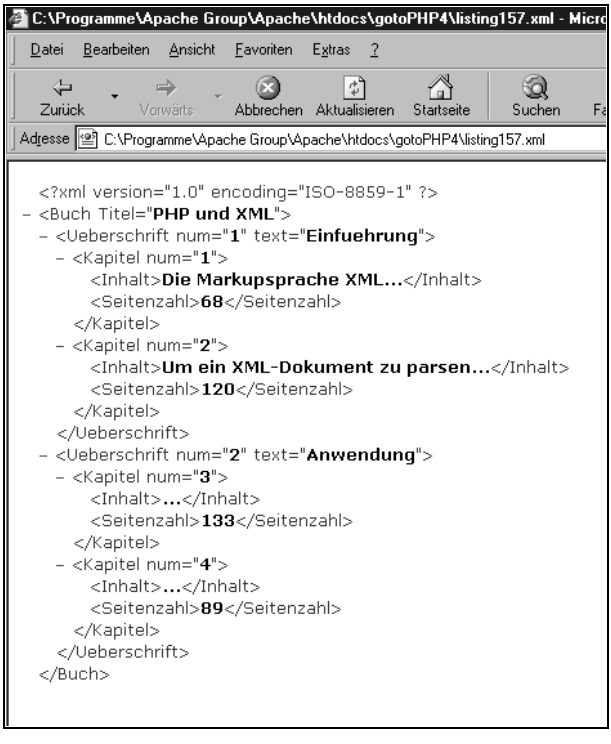


Abbildung 15.2: XML im Microsoft Explorer

Der Browser stellt die Struktur des Dokuments als einen aufklappbaren Baum dar, der über die Plus- und Minuszeichen von den Tags gesteuert werden kann. So ist es möglich, Teile des Baums auszublenden, um beispielsweise eine besseren Überblick zu gewinnen oder nur bestimmte Teile des Dokuments zu betrachten. Zusätzlich werden Tags und Daten durch die Darstellung farblich unterschieden, so dass die Orientierung leicht fällt.

Eine Interpretation der Daten im Browser findet allerdings nicht statt, auch wenn es auf den ersten Blick den Eindruck erweckt. Um die Daten korrekt interpretieren zu können, fehlen dem Explorer Daten zu den einzelnen Tags, die angeben, wie welcher Punkt dargestellt werden soll. Solange diese Informationen fehlen, gibt der Browser von Microsoft die Daten in ihrer Struktur aus.

Leider fehlt dem Browser von Netscape zur Zeit noch diese Flexibilität, da er ohne Darstellungsanweisungen lediglich die Daten unformatiert ausgibt. Da uns zur Zeit weniger die Darstellungsebene von XML interessiert, ist diese Software also fürs Erste uninteressant.

15.4.3 Sonderzeichen verwenden

Standardmäßig gehen die meisten XML-Parser davon aus, dass das XML-Dokument im UTF-8 Standard codiert worden ist. Das bedeutet, dass der Parser nur mit den ersten 128 Zeichen des ASCII-Zeichensatzes rechnet. Stößt er auf ein unerwartetes Zeichen, wird er mit einer Fehlermeldung den Vorgang beenden.

Um nicht auf Sonderzeichen verzichten zu müssen, kann über die Verarbeitungsanweisung <?xml ?> das Attribut »encoding« gesetzt werden, das dem Parser mitteilt, welcher Zeichensatz verwendet wird. Die Sonderzeichen vieler verschiedener Sprachen sind im Laufe der Zeit in diversen ISO-Normen festgelegt worden. Die westeuropäischen Sprachen, die alle gängigen Sonderzeichen der deutschen, französischen, spanischen und italienischen Sprache abdecken, sind beispielsweise in der ISO-Norm 8859-1 festgelegt.

Die folgende Tabelle zeigt eine Übersicht über die wichtigsten Zeichensätze in XML-Dokumenten:

ISO-Standard	Zeichensatz
ISO-8859-1	Westeuropa, Lateinamerika (Latin-1)
ISO-8859-2	Osteuropa (Latin-2)
ISO-8859-3	Südeuropa (Latin-3)
ISO-8859-4	Skandinavien, Baltikum (Latin-4)
ISO-8859-5	Kyrillisch
ISO-8859-6	Arabisch
ISO-8859-7	Griechisch
ISO-8859-8	Hebräisch
ISO-8859-9	Türkisch
ISO-8859-10	Lappländisch
ISO-2022-JP	Japanisch

Tabelle 15.2: Zeichensätze

Der jeweils verwendete Zeichensatz muss im <?xml?>-Tag mit dem Attribut »encoding« bekannt gemacht werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Für unsere Breitengrade ist der ISO-Standard ISO-8859-1 die beste Wahl und wird alle folgenden Beispiele begleiten.

15.4.4 Kommentare in XML

XML-Code kann wie fast jede Sprache mit Kommentaren versehen werden, die das Lesen und Bearbeiten in der Zukunft erleichtern sollen. Mit Kommentaren können Sie anderen Programmierern die Arbeit vereinfachen und Teile eines XML-Dokuments innerhalb des Quelltextes quasi »vor Ort« erklären. Kommentare in XML werden wie in HTML 4.0 eingefügt:

```
<!-- Ich bin ein Kommentar  -->
```

Nach einer geöffneten spitzen Klammer folgen ein Ausrufezeichen und dann zwei Gedankenstriche, die den folgenden Kommentar einleiten. Abgeschlossen wird der Kommentar über zwei weitere Gedankenstriche gefolgt von einer schließenden spitzen Klammer.

Alle Angaben innerhalb dieser beiden Klammern werden vom Parser ignoriert. Ein Kommentar kann sich über mehrere Zeilen erstrecken, solange er sich zwischen der öffnenden und schließenden Klammer befindet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
   Ein Kommentar zum Dokument:
   Das folgende Dokument beschreibt plakativ den Aufbau
   einiger Kapitel.
-->
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
         <Seitenzahl>68</Seitenzahl>
      </Kapitel>
   <!-- Hier endet das erste Kapitel -->
      <Kapitel num="2">
         <Inhalt>Um ein XML-Dokument zu parsen...</Inhalt>
         <Seitenzahl>120</Seitenzahl>
      </Kapitel>
   <!-- Hier endet das zweite Kapitel -->
   </Ueberschrift>
   <Ueberschrift num="2" text="Anwendung">
      <Kapitel num="3">
         <Inhalt>...</Inhalt>
         <Seitenzahl>133</Seitenzahl>
      </Kapitel>
   <!-- Hier endet das dritte Kapitel -->
      <Kapitel num="4">
         <Inhalt>...</Inhalt>
         <Seitenzahl>89</Seitenzahl>
      </Kapitel>
   <!-- Hier endet das vierte Kapitel -->
   </Ueberschrift>
</Buch>
```

Listing 15.2: Kommentare

Das Listing zeigt das erste XML-Beispiel mit einigen Kommentaren. Im Microsoft Explorer werden Kommentare grau dargestellt.

15.4.5 Festlegung einer Grammatik – die DTD

Alle bisherigen Beispiele für XML-Dokumente haben die Gemeinsamkeit, dass sie zwar Daten enthalten und zur Verfügung stellen, aber der Aufbau in keinem Regelwerk definiert worden ist. Die Tags definieren sich automatisch in dem Augenblick, in dem der Parser auf diesen Tag stößt. Theoretisch ist dieses Vorgehen möglich, sollte aber in der Praxis präziser gehandhabt werden.

XML stellt die Möglichkeit zur Verfügung Regeln zu definieren, die bestimmte Anforderungen an XML-Dokumente stellen, damit sie für einen bestimmten Parser gültig sind. Die Festlegung solcher Regeln trägt dazu bei, dass Dokumente einheitlich und in einer sinnvollen Form erstellt werden. So wird sichergestellt, dass bei der Interpretation von Daten keine Misverständnisse auftreten können.

Die Gesamtheit aller Regeln – die Grammatik – wird in der so genannten *Document Type Definition* (DTD) gespeichert, die im Kopf des XML-Dokuments zu finden ist. In dieser DTD wird beispielsweise festgelegt, welche Tags in diesem Dokument gültig sind und welche Attribute sie haben dürfen. Gleichzeitig wird die Struktur des XML-Dokuments bestimmt, indem für jedes Element die gültige Datenform vorgegeben wird.

Tags definieren

Die Document Type Definition wird direkt nach den PIs im Kopf eines XML-Dokuments angegeben. Die Definition beginnt immer mit einer sich öffnenden spitzen Klammer gefolgt von einem Ausrufezeichen und dem Schlüsselwort DOCTYPE. Nach diesem Einführungsprocedere folgen der eigentliche Name der Definition und ein Paar eckige Klammern, die den Inhalt der DTD umschließen. Abgeschlossen wird das Regelwerk von einer weiteren sich schließenden spitzen Klammer.

```
<!DOCTYPE name [
   ...
]>
```

Die Strukturierung der Informationen in einem XML-Dokument kann prinzipiell in zwei einfache Bereiche unterschieden werden. Auf der einen Seite die Menge der konkreten Daten, die im XML-Dokument strukturiert wird, und auf der anderen Seite die abstrakten Einheiten, die für die Gruppierung und Anordnung der Daten zuständig sind.

Bereich	Beispiel
abstrakte Einheiten (Tags)	<Kapitel> </Kapitel>
konkrete Daten	Zahlen, Buchstaben etc.

Tabelle 15.3: Informationsstrukturierung

Abstrakte Einheiten werden in XML als Elemente bezeichnet und beschreiben alle Tags, die für die Strukturierung der Daten im Dokument zuständig sind. Die konkreten Daten müssen immer zwischen zwei Elementen stehen, die ihren Datenwert repräsentieren. XML unterscheidet zwei verschiedene Arten von Elementen, die in einem Dokument auftauchen können:

- Datenelemente, die konkrete Daten enthalten
- Containerelemente, die selbst weitere Unterlemente enthalten

Daten- und Containerelemente können gemischt werden, indem sowohl konkrete Daten als auch weitere Unterelemente zugelassen werden. In der Praxis ist von diesem Vorgehen allerdings abzuraten, da so nicht immer eine eindeutige Zuordnung möglich ist.

Datenelemente müssen in der DTD besonders gekennzeichnet werden, damit klar ist, dass dieses Element eine Zeichenfolge enthalten darf. Die Deklaration eines solchen Elements sieht folgendermaßen aus:

```
<!ELEMENT Inhalt (#PCDATA)>
```

Jede Definition eines Elements beginnt mit einem Ausrufezeichen und dem Schlüsselwort ELEMENT gefolgt vom Namen dieser Definition. In Klammern folgt das gültige Format für dieses Element, welches in diesem Fall vom Typ PCDATA ist. Das Zeichen # gibt an, dass es sich hierbei um ein Schlüsselwort handelt, nämlich die Abkürzung für *parsed character data*. Elemente, die mit über dieses Format definiert wurden, dürfen konkrete Daten in Form von Buchstaben oder Zahlen enthalten, die in der jeweiligen ISO-Norm festgelegt wurden.

Im Gegensatz zu den Datenelementen enthalten Containerelemente keine konkreten Daten, sondern weitere Unterelemente, die das Dokument weiter strukturieren.

```
<!ELEMENT Ueberschrift (Kapitel)>
```

Anstelle des Schlüsselwortes PCDATA erscheint jetzt der Name des zulässigen Unterelements, das für dieses Element gültig ist. Diese Definition legt fest, dass alle Elemente »Ueberschrift« nur das Element »Kapitel« enthalten dürfen.

Es ist möglich, für ein Element mehrere zulässige Unterelemente zu definieren. Alle Elemente müssen durch Kommata in der Klammer getrennt werden.

```
<!ELEMENT Kapitel (Inhalt, Seitenzahl)>
```

Das Element Kapitel darf jetzt sowohl das Unterelement Inhalt als auch Seitenzahl enthalten. Beide sind für diesen Tag jetzt gültig.

Eigenschaften von Elementen

Die Elementdefinition für den Tag »Kapitel« weist in seiner jetzigen Form den Status obligatorisch für die Unterelemente »Inhalt« und »Seitenzahl« auf. Das bedeutet, dass jedes Element »Kapitel« ein, aber nicht mehrere Unterelemente von jeweils einem Typ haben darf. Beinhaltet der Tag »Kapitel« beispielsweise nicht den Untertag »Inhalt«, ist das XML-Dokument nicht gültig. Dasselbe gilt auch für die Situation, in der ein Untertag doppelt auftritt.

Wenn man die Gliederung eines Buches betrachtet, dann ist diese Festlegung für das Element »Kapitel« durchaus sinnvoll, denn zwei verschiedene Angaben zur Seitenzahl oder zum Inhalt sind sicher blödsinnig. Anders sieht es allerdings beim Element »Ueberschrift« aus, das durchaus mehrere Unterelemente vom Typ Kapitel enthalten darf. Die jetzige Formulierung der DTD verbietet dies allerdings.

Möchte man die Eigenschaft eines Elements dahingehend verändern, dass mehrere Unterlemente vom gleichen Typ zugelassen werden, dann geschieht das über verschiedene Steuerzeichen nach den einzelnen Namen.

```
<!ELEMENT Ueberschrift (Kapitel+)>
```

Das Pluszeichen nach dem Namen des möglichen Unterelements gibt an, dass der Tag mindestens einmal, aber auch beliebig oft auftreten kann. Insgesamt gibt es vier verschiedene Möglichkeiten die Eigenschaft eines Unterelements anzugeben:

Eigenschaft	Beschreibung
`name`	Das Element muss genau einmal auftreten
`name?`	Das Element kann einmal auftreten, kann aber auch weggelassen werden.
`name+`	Das Element muss mindestens einmal, kann aber auch beliebig oft auftreten.
`name*`	Das Element kann beliebig oft auftreten, kann aber auch komplett weggelassen werden.

Tabelle 15.4: Eigenschaften von Elementen

Die Codierung der Eigenschaften richtet sich in groben Zügen nach den Möglichkeiten der regulären Ausdrücke, die mit ganz ähnlichen Methoden arbeiten.

Die komplette DTD für das oben beschriebene XML-Dokument eines Buches würde nach den bisherigen Gesichtspunkten also so aussehen:

```
<!DOCTYPE name [
   <!ELEMENT Buch (Ueberschrift+)>
   <!ELEMENT Ueberschrift (Kapitel+)>
   <!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
   <!ELEMENT Inhalt (#PCDATA)>
   <!ELEMENT Seitenzahl (#PCDATA)>
]>
```

Listing 15.3: DTD für die Gliederung eines Buches

Diese Document Type Definition legt fest, dass die Elemente »Ueberschrift« und »Kapitel« im jeweiligen Kontext mindestens einmal, aber beliebig oft auftreten können. Diese Angabe ist für ein Buch sicherlich sinnvoll, da ein Buch bestimmt über mehrere Überschriften und Kapitel verfügt. Die Angabe der Seitenzahl wurde als optional definiert. Der Anwender kann entscheiden, ob er diese Angabe machen will oder nicht.

Konjunktoren für Elemente

Eine weitere Möglichkeit die Struktur der Unterlemente eines Tags zu definieren ist der Einsatz von Konjunktoren aus der Logik, die angeben in welcher Reihenfolge oder in welcher Kombination die Elemente auftreten dürfen. In den bisherigen Beispielen haben wir nur das Komma für die Verbindung von Elementen verwendet, das implizit die Reihenfolge festlegt.

```
<!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
```

Diese Angabe bestimmt, dass der Untertag Seitenzahl innerhalb eines Kapitels erst nach dem Tag »Inhalt« auftreten darf. Eine umgekehrte Angabe der Daten ist nicht gültig. Nebem dem Komma gibt es einen weiteren Konjunktor, der in der Lage ist Alternativen auszudrücken.

```
<!ELEMENT Bestellung (Kundennummer | Adresse , Waren)>
```

Das Zeichen | (senkrechter Strich) definiert ein logisches »Entweder ... oder« und stellt damit die Wahl des Unterelements frei. Das Beispiel zeigt die Definition eines Elements mit dem Namen »Bestellung«, das drei Unterlemente zulässt. Die ersten beiden Möglichkeiten »Kundennummer« und »Adresse« stehen sich alternativ gegenüber, da eine der beiden Angaben für die Identifikation des Kunden ausreicht. Es ist denkbar, dass das Element »Adresse« in weitere Unterelemente gegliedert ist.

In SGML, dem Vorgänger von XML, gibt es einen weiteren Konjunktor (&), der die Reihenfolge der Elemente freistellt. Für XML wurde diese Möglichkeit gestrichen, kann aber bei Bedarf sehr leicht nachgebaut werden:

```
a & b = (a, b) | (b, a)
```

Der Einsatz von Klammern in einem Inhaltsmodell für Elemente erlaubt es, so genannte Untermodelle zu definieren, die sowohl mit Konjunktoren verknüpft als auch mit Eigenschaftsangaben ausgestattet werden können. So ist es möglich, mehrere Unterelemente mit denselben Eigenschaften zusammenzufassen.

```
<!ELEMENT a ((b , c)* | d+)>
```

Die Beispieldefinition legt die Unterelemente für das Element a in zwei verschiedenen Möglichkeiten fest: Entweder werden die Elemente a und b in dieser Reihenfolge kein Mal oder beliebig oft verwendet oder Element d tritt mindestens einmal auf.

Attribute für Tags definieren

Im letzten Kapitel haben Sie bereits gesehen, wie Attribute für Elemente genutzt werden, um weitere Informationen in einem Datenstrukturbaum zu speichern. Genau wie für Tags kann in einer DTD auch die Verwendung von Attributen definiert werden. Im Beispiel zur Struktur eines Buches wurden verschiedene Tags mit Attributen versehen:

```
<Ueberschrift num="1" text="Einfuehrung">
<Kapitel num="1">
```

Um ein Attribut für ein Element in der DTD zu deklarieren, muss es über das Schlüsselwort ATTLIST bekannt gemacht werden. Die Deklaration gibt an, für welches Element das Attribut eingesetzt werden soll und welchen Namen es besitzt. Zusätzlich muss eine Liste von möglichen Werten angegeben werden, die für das Attribut Gültigkeit haben. Abgeschlossen wird die Deklaration von einem Schlüsselwort, das den Status des Attributs bestimmt.

```
<!ATTLIST Kapitel num (1|2|3|4|5|6) #REQUIRED>
```

Das neue Attribut erhält durch diese Deklaration den Namen num, die möglichen Werte 1 bis 6 sowie die Statusangabe REQUIRED, was besagt, dass dieses Attribut im Element Kapitel immer mit einem Wert versehen sein muss. Die möglichen Werte des Attributs werden in Klammern nach dem Namen angegeben und durch senkrechte Striche (oder) voneinander getrennt.

Neben dem Schlüsselwort REQUIRED kann ein Attribut auch mit als IMPLIED definiert werden, was die Angabe dieses Wertes optional macht.

```
<!ATTLIST Kapitel num (1|2|3|4|5|6) #IMPLIED>
```

Der User kann nun entscheiden, ob er diesen Parameter setzen möchte oder nicht. Ein typisches Beispiel für eine solche Angabe ist der Parameter »border« im Image-Tag von HTML. Über ihn kann die Rahmenbreite des Bildes festgelegt werden. Fehlt der Tag, wird kein Rahmen für das Bild ausgegeben.

Mehrere Attribute für ein Element

Selbstverständlich ist die Verwendung von Attributen nicht auf eines pro Element beschränkt. Die Kombination von mehreren Attributen ermöglicht es, Elemente genauer zu identifizieren und mehr Informationen zu verwenden.

Jedes weitere Attribut eines bestimmten Elements wird im selben ATTLIST-Tag untergebracht, wie die übrigen Attribute. Zur besseren Übersicht empfiehlt es sich, alle Attribute untereinander zu schreiben.

```
<!ATTLIST Ueberschrift
   num (1|2|3|4)  #REQUIRED
   text (Einfuehrung|Anwendung|Index) #IMPLIED
>
```

Der Ausschnitt einer DTD zeigt, wie die Attribute »num« und »text« für das Element »Ueberschrift« definiert werden. Das Attribut »text« ist als IMPLIED angegeben, die Verwendung ist also optional.

Die komplette DTD für die Beispielstrukturierung eines Buches mit dem eigentlichen XML-Dokument sieht also so aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Die DTD -->
<!DOCTYPE name [
   <!ELEMENT Buch (Ueberschrift+)>
   <!ELEMENT Ueberschrift (Kapitel+)>
   <!ATTLIST Ueberschrift
      num (1|2|3|4)  #REQUIRED
      text (Einfuehrung|Anwendung|Index) #IMPLIED
   >
   <!ATTLIST Kapitel num (1|2|3|4|5|6) #REQUIRED>
   <!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
   <!ELEMENT Inhalt (#PCDATA)>
   <!ELEMENT Seitenzahl (#PCDATA)>
]>
<!-- Hier beginnt das eigentliche XML-Dokument -->
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
         <Seitenzahl>68</Seitenzahl>
      </Kapitel>
      <Kapitel num="2">
         <Inhalt>Um ein XML-Dokument zu parsen...</Inhalt>
         <Seitenzahl>120</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
   <Ueberschrift num="2" text="Anwendung">
      <Kapitel num="3">
         <Inhalt>...</Inhalt>
         <Seitenzahl>133</Seitenzahl>
      </Kapitel>
      <Kapitel num="4">
         <Inhalt>...</Inhalt>
         <Seitenzahl>89</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
</Buch>
```

Listing 15.4: DTD mit XML-Dokument

Im Microsoft Explorer wird das Dokument so dargestellt:

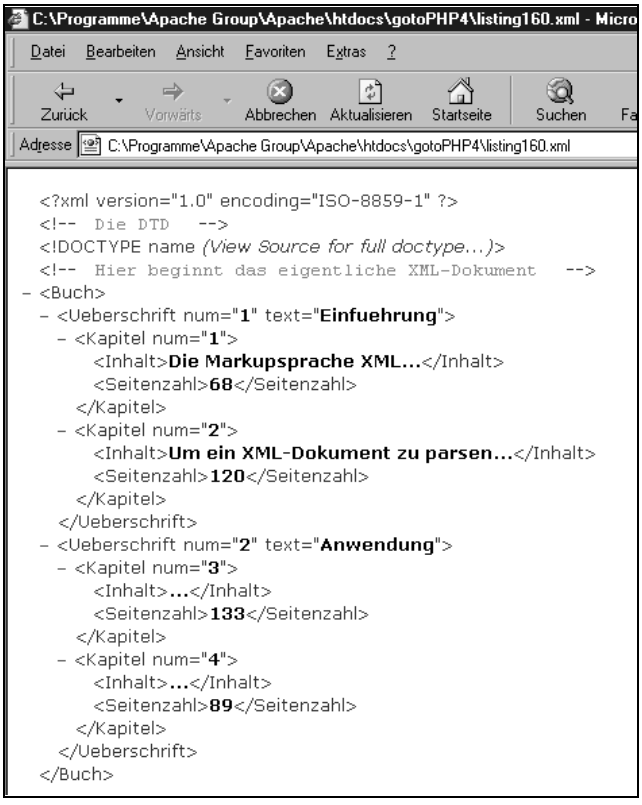


Abbildung 15.3: XML mit DTD im Explorer

Anmerkungen zum Microsoft Explorer

Der Microsoft Explorer kontrolliert zwar die DTD auf die syntaktische Korrektheit der Angaben, stellt aber nicht sicher, ob das XML-Dokument auch den Vorgaben entspricht. Der Browser dient also lediglich zur Kontrolle auf eventuelle Tippfehler im Code.

Ein Bug im Microsoft Explorer 5 verhindert, dass korrekte XML-Dokumente, die sowohl das Schlüsselwort REQUIRED als auch IMPLIED verwenden, nicht dargestellt werden können. Das Programm gibt einen Fehler an, obwohl es richtiges XML ist. In den jüngeren Versionen vom Explorer ist das Problem behoben.

Weitere Möglichkeiten der Attribut-Deklaration

XML stellt die Möglichkeit zur Verfügung für optionale Attribute Standardwerte zu definieren, die automatisch verwendet werden, wenn ein Attribut nicht gesetzt wird. Diese Technik findet in HTML beispielsweise beim Tag <H1> Anwendung:

```
<H1>Überschrift</H1>
```

Wird der HTML-Tag ohne weitere Parameter verwendet, dann geht der Browser automatisch davon aus, dass ALIGN auf den Wert »left« gesetzt wurde. Die Überschrift wird also linksbündig ausgegeben. In der XML-Definition dieses Tags wurde für den Parameter ALIGN ein Standardwert angegeben, der immer den Wert »left« besitzt.

Eine solche Voreinstellung kann für jedes Attribut angegeben werden. Die Festlegung wird in der DTD direkt bei der jeweiligen Definition vorgenommen, und zwar gleich nach den möglichen Werten des Attributs.

```
<!ATTLIST Kapitel num (1|2|3|4|5|6) "1" #IMPLIED>
```

Der Standardwert für das Attribut wird in Anführungszeichen vor das Status-Schlüsselwort des Attributs gesetzt. Das Beispiel definiert für das Attribut num den Wert 1 als Voreinstellung. Wenn dieses Attribut im XML-Dokument weggelassen wird, geht der Parser automatisch davon aus, dass num gleich 1 ist.

Ein wichtiger Spezialfall bei der Deklaration von Attributwerten ist die Möglichkeit der Vergabe von eindeutigen Identifikatoren für ein Element im Baum. Der dafür vorgesehene Typ eines Attributs ist ID.

```
<!ATTLIST Kapitel num ID #REQUIRED>
```

Erhält ein Attribut diesen Typ zugewiesen, dann muss der Wert dieses Attributs einen Namen zugewiesen bekommen, der im gesamten XML-Dokument kein zweites Mal verwendet wurde. Pro Attribut darf also nur einmal derselbe Wert auftreten. Wird demnach das Kapitel-Element wie folgt definiert

```
<Kapitel num="1">
```

dann wird jeder Parser einen Fehler auswerfen, wenn derselbe Wert ein zweites Mal bei einem als ID definierten Attribut verwendet wird.

Der Attributstyp ID erlaubt es, Elemente eindeutig identifizieren zu können, indem man Teilmengen aus den Daten des XML-Dokuments bildet, die genau diese festgelegte Eigenschaft besitzen. Dabei ist der Typ des Elements unwichtig, da ID-Attribute nur ein einziges Mal verwendet werden dürfen.

Abkürzungen

Innerhalb der Sprache XML existiert die Möglichkeit oft wiederkehrende Texte abzukürzen und durch ein Kürzel zu ersetzen. Diese Abkürzungen werden Entities genannt und werden in der Praxis häufig eingesetzt. Die Erstellung einer Abkürzung ist relativ einfach zu handhaben:

```
<!ENTITY text "Dies ist eine Abkürzung!">
```

Das Schlüsselwort ENTITY leitet die Abkürzungsdefinition ein gefolgt vom Namen der Abkürzung und dem eigentlichen Inhalt. Im XML-Code wird dann nur noch die Abkürzung aufgerufen, die nun den Text an die entsprechende Stelle transferiert.

```
&text;
```

Der Aufruf der Abkürzung erfolgt über das kaufmännische Und (&), gefolgt vom Namen und einem Semikolon. Es hat sich eingebürgert, dass Abkürzungen in der DTD erst nach den Elementdefinitionen deklariert werden. In der Praxis sieht der Einsatz der Abkürzungen so aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Die DTD -->
<!DOCTYPE name [
   <!ELEMENT Buch (Ueberschrift+)>
   <!ELEMENT Ueberschrift (Kapitel+)>
   <!ATTLIST Ueberschrift
      num (1|2|3|4)  #REQUIRED
      text (Einfuehrung|Anwendung|Index) #IMPLIED
   >
   <!ATTLIST Kapitel num (1|2|3|4|5|6) #REQUIRED>
   <!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
   <!ELEMENT Inhalt (#PCDATA)>
   <!ELEMENT Seitenzahl (#PCDATA)>
   <!ENTITY text "Dies ist eine Abkürzung!">
   <!ENTITY text2 "Dies ist noch eine Abkürzung!">
]>
<!-- Hier beginnt das eigentliche XML-Dokument -->
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
```

```
         <Inhalt>&text;</Inhalt>
         <Seitenzahl>68</Seitenzahl>
      </Kapitel>
      <Kapitel num="2">
         <Inhalt>&text2;</Inhalt>
         <Seitenzahl>120</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
</Buch>
```

Listing 15.5: Arbeit mit Abkürzungen

Die Ausgabe im Microsoft Explorer sieht so aus:

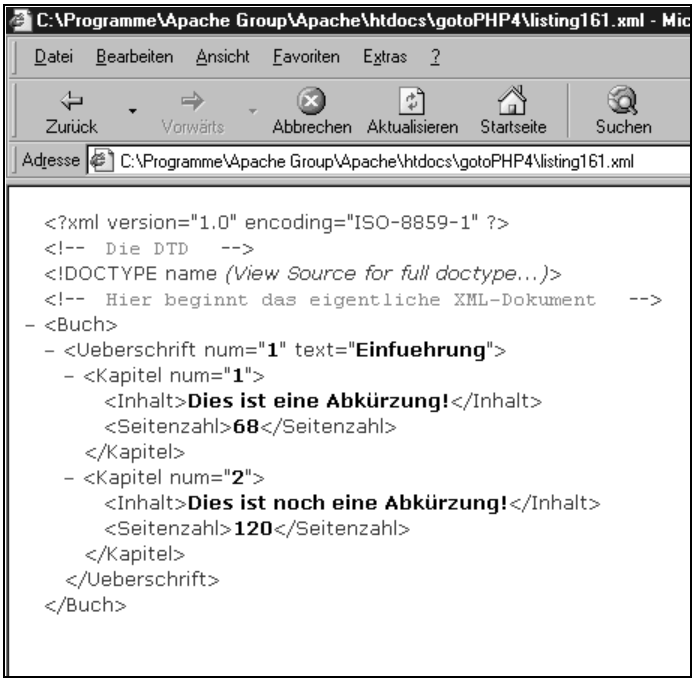


Abbildung 15.4: Arbeit mit Abkürzungen

Wohlgeformte und gültige Dokumente

Bei XML wird zwischen wohlgeformten (engl.: well-formed) und gültigen (engl.: valid) Dokumenten unterschieden. In den letzten Kapiteln haben wir sowohl gültige als auch wohlgeformte Dokumente benutzt und

erstellt, ohne dass ich darauf aufmerksam machen musste. Der Unterschied ist also rein formeller Natur und eigentlich schnell erklärt.

Ein Dokument wird als wohlgeformt bezeichnet, wenn es keine DTD besitzt, aber jeder Tag durch einen Endtag abgeschlossen ist. Das Dokument muss syntaktisch einwandfrei sein und darf keine Fehler enthalten, die gegen die XML-Konventionen verstoßen. Das folgende Dokument ist wohlgeformt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
      </Kapitel>
    </Ueberschrift>
</Buch>
```

Ein Dokument wird als gültig bezeichnet, wenn es eine DTD als Vorlage hat und der Inhalt des XML-Dokuments den Regeln der DTD entspricht. Alle Tags müssen so eingesetzt werden, wie es die DTD vorsieht. Das folgende Dokument ist gültig:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE name [
   <!ELEMENT Buch (Ueberschrift+)>
   <!ELEMENT Ueberschrift (Kapitel+)>
   <!ATTLIST Ueberschrift
      num (1|2|3|4)  #REQUIRED
      text (Einfuehrung|Anwendung|Index) #IMPLIED
   >
   <!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
   <!ATTLIST Kapitel num (1|2|3|4|5|6) #REQUIRED>
   <!ELEMENT Inhalt (#PCDATA)>
]>
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
      </Kapitel>
    </Ueberschrift>
</Buch>
```

Jedes gültige Element muss auch gleichzeitig ein wohlgeformtes Element sein, da syntaktische Fehler im Dokument den Einsatz der DTD verhindern. Umgekehrt muss aber ein wohlgeformtes Dokument nicht gültig sein, da eine DTD nicht vorausgesetzt wird.

Interne und externe DTD

Bei allen bisherigen Beispielen befinden sich die DTDs innerhalb des Dokuments, in dem auch die konkreten Daten gespeichert sind. Diese Organisation wird als interne DTD bezeichnet, weil sich die Daten und die formalen Strukturierungsanweisungen in derselben Datei befinden. Für kleine Anwendungen ist eine interne DTD vollkommen ausreichend, aber für größere Projekte wird diese Technik schnell unübersichtlich.

Um ein Chaos zwischen Daten und DTD zu verhindern, sollten Sie diese trennen und die DTD in einer eigenen Datei speichern. In der Datendatei wird an der Stelle, wo vorher die interne DTD stand, ein Verweis auf die externe Datei gemacht:

```
<!DOCTYPE name SYSTEM "datei.dtd">
```

Der Verweis beginnt genau wie eine interne DTD mit dem Schlüsselwort DOCTYPE, gefolgt vom Namen der Definition. Das Schlüsselwort SYSTEM teilt dem Parser mit, dass die DTD, die geladen werden soll, nur für das lokale System (den Rechner) gelten soll. Danach folgt der Name der Datei in Anführungszeichen (datei.dtd).

Neben dem Schlüsselwort SYSTEM existiert noch das Schlüsselwort PUBLIC, das für den Einsatz von öffentlichen DTDs verwendet wird. Es sollte nur verwendet werden, wenn eine allgemein zugängliche DTD eingesetzt wird, die beispielsweise von einer Webseite des W3-Konsortiums heruntergeladen werden kann.

Die Angabe der Zieldatei mit der gespeicherten DTD kann entweder als relativer oder als absoluter Pfad angegeben werden. Im oberen Beispiel ist die Angabe relativ zu verstehen, da keine Laufwerksbezeichnung abgegeben wurde. Der Parser versucht also die Datei aus demselben Verzeichnis zu laden, in dem das XML-Dokument gespeichert ist. Eine absolute URL gibt die Position des Dokuments immer vom Root-Verzeichnis des Systems (oder des Laufwerks) aus an.

```
<!DOCTYPE name SYSTEM "c:\xml\dtd\datei.dtd">
```

Je nach Betriebssystem muss auf die richtige Syntax der Verzeichnisangaben geachtet werden. Das folgende Beispiel zeigt eine absolute Pfadangabe auf UNIX:

```
<!DOCTYPE name SYSTEM "/usr/xml/datei.dtd">
```

Neben Laufwerksangaben ist es auch möglich, komplette URLs für die DTD-Datei anzugeben. Der Parser wird die benötigten Informationen automatisch von der angegebenen Adresse herunterladen.

```
<!DOCTYPE name PUBLIC "http://www.kulturbrand.de/xml/dtd/datei.dtd">
```

Dieser Zugriff ist natürlich nur möglich, wenn ein Internetzugang besteht.

Die externe DTD unterscheidet sich nicht von der internen DTD, mit der Ausnahme, dass auf den einleitenden Tag DOCTYPE verzichtet wird. Dieser wird schon durch den Verweis auf das Dokument in der XML-Datei gesetzt und darf nicht noch mal auftauchen.

Das folgende Beispiel zeigt zum Beispiel das XML-Dokument der Strukturierung eines Buches mit einer externen DTD:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Der DTD-Verweis -->
<!DOCTYPE myDTD SYSTEM "listing162.dtd">
<!-- Hier beginnt das eigentliche XML-Dokument -->
<Buch>
   <Ueberschrift num="1" text="Einfuehrung">
      <Kapitel num="1">
         <Inhalt>Die Markupsprache XML...</Inhalt>
         <Seitenzahl>68</Seitenzahl>
      </Kapitel>
      <Kapitel num="2">
         <Inhalt>Um ein XML-Dokument zu parsen...</Inhalt>
         <Seitenzahl>120</Seitenzahl>
      </Kapitel>
   </Ueberschrift>
   <Ueberschrift num="2" text="Anwendung">
      <Kapitel num="3">
         <Inhalt>...</Inhalt>
         <Seitenzahl>133</Seitenzahl>
      </Kapitel>
      <Kapitel num="4">
         <Inhalt>...</Inhalt>
         <Seitenzahl>89</Seitenzahl>
```

```
      </Kapitel>
   </Ueberschrift>
</Buch>
```

Listing 15.6: Ein XML-Dokument mit externer DTD

Die externe DTD-Datei »listing162.dtd« sieht so aus:

```
<!ELEMENT Buch (Ueberschrift+)>
<!ELEMENT Ueberschrift (Kapitel+)>
<!ATTLIST Ueberschrift
   num (1|2|3|4)  #REQUIRED
   text (Einfuehrung|Anwendung|Index) #IMPLIED
>
<!ATTLIST Kapitel num (1|2|3|4|5|6) #REQUIRED>
<!ELEMENT Kapitel (Inhalt, Seitenzahl?)>
<!ELEMENT Inhalt (#PCDATA)>
<!ELEMENT Seitenzahl (#PCDATA)>
```

Listing 15.7: Eine externe DTD

Die Trennung zwischen XML-Dokument und den Strukturanweisungen der DTD schafft eine bessere Übersicht und erleichtert die spätere Wartung des Codes. Eine externe DTD erlaubt es außerdem, von verschiedenen Dokumenten auf dieselbe DTD zuzugreifen und sie zu nutzen. So ist es möglich, die gleichen Vorlagen in verschiedenen Dokumenten zu verwenden, ohne die verwendete DTD immer wieder zu kopieren.

Ich kann nur empfehlen immer externe zu verwenden, weil sie eine ganze Reihe von Vorteilen bieten.

15.5 XML und PHP

PHP bietet zwei verschiedene Wege ein XML-Dokument zu interpretieren, um auf die gespeicherten Daten zugreifen zu können. Die erste Möglichkeit, ein ereignisgestützter Parser, ist automatisch in der aktuellen PHP Distribution enthalten. Die zweite Möglichkeit, der Zugriff auf die Daten über das so genannte DOM-Model, muss über ein externes Modul nachinstalliert werden. Mehr Informationen dazu finden Sie in einem der folgenden Kapitel.

PHP unterstützt mit einer Reihe von Funktionen die so genannte *Simple API for XML* (SAX), die das eventgesteuerte Parsen von XML-Dokumenten erlaubt. Die erste Version der SAX wurde am 11. Mai 1998 von den Mitgliedern der XML-DEV Mailing List entwickelt. Inzwischen ist diese

Standardschnittstelle (Application Programmers Interface: API) neben DOM eine der wichtigsten Möglichkeiten im Umgang mit XML.

Der Ansatz von SAX ist grundsätzlich unterschiedlich von DOM. Hier wird kein hierarchisches Baummodel des Dokuments aufgebaut, sondern eine Sequenz von Ereignissen erzeugt, die bearbeitet werden. Der Vorteil hiervon gegenüber DOM ist, dass kein vorgegebenes Objektmodell verwendet werden muss, sondern ein eigenes entwickelt werden kann. Ein weiterer Punkt ist der weitaus schonendere Umgang mit den Systemressourcen, da im Gegensatz zu DOM nicht das komplette Dokument in den Speicher geladen wird.

SAX ist »event based«. Das bedeutet, dass der Parser, wenn er ein bestimmtes XML-Konstrukt (beispielsweise ein Element, einen Kommentar oder eine Processing Instruction) liest, ein Ereignis auslöst. Auch bei Fehlern oder Warnungen löst der Parser ein Ereignis aus. Diese Ereignisse können von einem Programmierer ganz individuell behandelt werden, indem jede Möglichkeit von einer bestimmten Funktion abgefangen wird. So kann die Applikation je nach Anforderung anders auf das XML-Dokument reagieren.

Um die Arbeitsweise von SAX zu verdeutlichen, sollen mit dem folgenden Beispiel die Arbeitsschritte eines Parsers demonstriert werden.

```
<Name>
   <Vorname>Dirk</Vorname>
   <Nachname>Ammelburger</Nachname>
</Name>
```

Dieses XML-Dokument wird jetzt von einem SAX-Parser analysiert. Der Parser würde beim Lesen folgende Ereignisse auslösen:

```
geöffnetes Element: "Name"
geöffnetes Element: "Vorname"
Character Data: "Dirk"
geschlossenes Element: "Vorname"
geöffnetes Element: "Nachname"
Character Data: "Ammelburger"
geschlossenes Element: "Nachname"
geschlossenes Element: "Name"
```

Wenn nun ein Ereignis auftritt, wird der entsprechende Eventhandler aufgerufen, der über eine Methode realisiert wird. Diese Methode wird vom Programmierer entsprechend den Anforderungen gestaltet.

Der SAX-Parser ist also nichts anderes als ein recht primitives Instrument um ein XML-Dokument auf bestimmte Tags und entsprechende Inhalte hin zu untersuchen. Im Gegensatz zum DOM wird der Inhalt des Dokumentes nicht gespeichert, sondern ist nach dem Auslösen des Event wieder »weg«. Es ist also die Aufgabe des Programmierers, den interessanten Inhalt in entsprechenden Variablen zu speichern.

Alle PHP-Funktionen, die mit der *Simple API for XML* arbeiten, beginnen mit dem Präfix xml_. Alle Funktionen für die Arbeit mit dem *Document Object Model* haben das Präfix `domxml_`.

15.5.1 Einen XML-Parser erstellen

Um mit einem PHP-Skript ein XML-Dokument parsen zu können, ist es im ersten Schritt notwendig, einen Parser zu erstellen. Diese Aufgabe wird durch die Funktion `xml_parser_create()` erledigt.

```
$xml_parser = xml_parser_create();
```

Die Funktion gibt im Erfolgsfall ein Handle auf den erstellten Parser zurück, ansonsten den Wert FALSE. Über einen optionalen Parameter kann die Codierung für den Parser festgelegt werden. Möglich sind drei Werte:

Wert	Beschreibung
ISO-8859-1	westeuropäische Sonderzeichen sind erlaubt (default).
US-ASCII	Das amerikanische ASCII-System
UTF-8	8 Bit-Codierung

Tabelle 15.5: Mögliche Kodierungen für den Parser

```
$xml_parser = xml_parser_create("US-ASCII");
```

Das Beispiel zeigt, wie der XML-Parser auf die Codierung US-ASCII festgelegt wird. Die Standardeinstellung ist ISO-8859-1. Alle weiteren Funktionen, die mit diesem Parser arbeiten sollen, müssen das zurückgegebene Handle übergeben bekommen.

Nachdem der Parser eingerichtet worden ist, wurde der erste Schritt getan. Im Folgenden müssen die Eventfunktionen definiert werden, die

der Parser aufrufen soll, wenn ein Ereignis eintritt. Zu diesem Zweck hat PHP folgende Konfigurationsfunktionen integriert:

Funktion	Beschreibung
`xml_set_element_handler()`	Diese Funktion definiert die Funktionen, die vom Parser aufgerufen werden, wenn ein XML-Element erreicht oder verlassen wird.
`xml_set_character_data_handler()`	Die Funktion definiert die Funktion, die vom Parser aufgerufen wird, wenn konkrete Daten geparsed werden.
`xml_set_processing_instruction_handler()`	Die Funktion definiert die Funktion, die vom Parser aufgerufen wird, wenn PIs im XML-Dokument geparst werden.
`xml_set_default_handler()`	Die Funktion definiert eine Funktion, die vom Parser aufgerufen wird, wenn Daten geparst werden, für die kein Ereignis zuständig ist.
`xml_set_unparsed_entity_decl_handler()`	Die Funktion definiert eine Ereignisfunktion, die bei nicht geparsten Einheiten aufgerufen wird.
`xml_set_notation_decl_handler()`	Die Funktion definiert die Ereignis-Routine, mit der Notations-Deklarationen behandelt werden.
`xml_set_external_entity_ref_handler()`	Die Funktion definiert die Funktion, die vom Parser aufgerufen wird, wenn eine externe Referenz geparst wird.

Tabelle 15.6: Ereignisbehandlungsroutinen in PHP

Auf den folgenden Seiten werde ich die wichtigsten Ereignisbehandlungsroutinen vorstellen, die für das Parsen eines XML-Dokuments unabdingbar sind.

Die Funktion xml_set_element_handler()

Mit der Funktion `xml_set_element_handler()` werden die Ereignisbehandlungsroutinen für Tag-Elemente in einem XML-Dokument definiert. Die Funktion macht zwei Funktionen bekannt, von denen eine für die Behandlung von Starttags und die andere für die Behandlung von Endtags angegeben wird. Die Funktion erwartet drei Parameter:

```
xml_set_element_handler($xml_parser, "startE", "endE");
```

Der erste Parameter übergibt das Handle des XML-Parsers, das durch die Funktion `xml_parser_create()` erstellt wurde. Der zweite Parameter ist ein String mit dem Namen der Funktion, die bei sich öffnenden Tags aufgerufen werden soll. Der dritte Parameter ist der Name der Funktion, die für schließende Tags zuständig sind.

Beide Funktionen müssen selbstverständlich über das Schlüsselwort `funktion` im PHP-Quelltext definiert werden, damit der Parser sie auch aufrufen kann. Beim Aufruf werden alle relevanten Informationen (Name des Tags, Attribute) als Parameter an die jeweilige Funktion weitergegeben, darum muss die Funktionsdeklaration in einem fest vorgeschriebenen Format erfolgen. Die Funktion für öffnende Tags muss drei Parameter akzeptieren:

```
function startE($xml_parser, $name, $attr)
{...}
```

- `$xml_parser` ist der Verweis auf das Handle zum Parser.
- `$name` ist der Name des Tag-Elements.
- `$attr` ist ein assoziatives Array mit den Attributen des Elements. Der Schlüssel sind die Attribute und die Elemente die Werte.

Die drei Werte stehen innerhalb des Parsers zur Verfügung und können beliebig eingesetzt werden, um mit den Informationen des XML-Dokuments zu arbeiten.

Die Funktion für schließende Tag-Elemente kommt mit zwei Parametern aus und muss wie folgt deklariert werden:

```
function endE($xml_parser, $name)
{...}
```

- `$xml_parser` ist der Verweis auf das Handle zum Parser
- `$name` ist der Name des Tag-Elements

Wenn innerhalb der Funktion `xml_set_element_handler()` eine oder beide Ereignisbehandlungsroutinen auf eine leere Zeichenkette gesetzt werden, dann ist die entsprechende Behandlungsroutine ausgeschaltet. Die Funktion gibt TRUE zurück, wenn das Setzen der Behandlungsfunktionen erfolgreich war. Wenn ein Fehler vorliegt, wird FALSE zurückgegeben.

Die Funktion xml_set_character_data_handler()

Über die Funktion `xml_set_character_data_handler()` wird die Behandlungsfunktion für Zeichendaten definiert. Die Funktion erwartet zwei Parameter:

```
xml_set_character_data_handler($xml_parser, "PCdata");
```

Der erste Parameter ist wie gehabt der Verweis auf das Parserhandle. Der zweite Parameter gibt die Funktion für die Auswertung von Zeichendaten – also alle Daten außerhalb von Tags – in einem XML-Dokument an. Die Zielfunktion muss zwei Parameter akzeptieren, um die Daten vom Parser übernehmen zu können:

```
function PCdata($xml_parser, $data)
{...}
```

- `$xml_parser` ist der Verweis auf das Handle zum Parser.
- `$data` speichert die Zeichendaten aus dem XML-Dokument, die bearbeitet werden sollen. Die Daten werden als String übergeben.

Die Behandlungsroutine kann durch die Übergabe einer leeren Zeichenkette ausgeschaltet werden. Im Erfolgsfall gibt die Funktion `xml_set_character_data_handler()` TRUE zurück, ansonsten FALSE.

Die Funktion xml_set_processing_instruction_handler()

Mit der Funktion `xml_set_processing_instruction_handler()` werden Funktionen für die Verarbeitung von PIs in einem XML-Dokument definiert. Prozessinformationen haben in XML immer folgendes Format:

```
<?ziel information?>
```

Immer wenn der Parser im Dokument auf eine Passage wie diese stößt, wird die Ereignisfunktion zur Bearbeitung dieser Anweisungen aufgerufen. Die Funktion erwartet zwei Parameter:

```
xml_set_processing_instruction_handler($xml_parser, "PIdata");
```

Der erste Parameter übergibt das Handle auf den XML-Parser, während der zweite Parameter den Namen der Eventfunktion bestimmt. Die Eventfunktion hat die Aufgabe die Angaben in den PIs eines XML-Dokuments umzusetzen. Die entsprechenden Informationen werden als Parameter übergeben:

```
function PIdata($xml_parser, $ziel, $information)
{...}
```

- `$xml_parser` ist der Verweis auf das Handle zum Parser
- `$ziel` übergibt den ersten Wert des PI-Elements und speichert die Zielinformation, für die die folgenden Angaben gelten
- `$information` enthält die Informationen über das zuvor definierte Ziel

Die Behandlungsroutine kann durch die Übergabe einer leeren Zeichenkette ausgeschaltet werden. Im Erfolgsfall gibt die Funktion TRUE zurück, ansonsten FALSE.

Die Funktion xml_set_default_handler()

Die Funktion `xml_set_default_handler()` setzt eine Standardbehandlungsroutine für die Bearbeitung aller nicht erfassten Ereignisse. Die Funktion wird immer dann aufgerufen, wenn der Parser keine andere Funktion für dieses Ereignis findet. Dazu gehören beispielsweise unerwartete Zeichen oder eine interne DTD. Die Funktion erwartet zwei Parameter:

```
xml_set_default_handler($xml_parser, "default_handler");
```

Neben dem Verweis auf den XML-Parser wird der Name der Funktion festgelegt, die im Fall eines »Standardereignisses« aufgerufen werden soll. Für die Umsetzung der Funktion müssen zwei Parameter berücksichtig werden:

```
function default_handler($xml_parser, $data)
{...}
```

- `$xml_parser` ist der Verweis auf das Handle zum Parser
- `$data` übergibt die Daten, die das Ereignis ausgelöst haben

Die Behandlungsroutine kann durch die Übergabe einer leeren Zeichenkette ausgeschaltet werden. Im Erfolgsfall gibt die Funktion TRUE zurück, ansonsten FALSE.

Ein kompletter Parser

Mit den vorgestellten Funktion ist es jetzt möglich, einen kompletten XML-Parser aufzusetzen:

```
<?
//Ereignisbehandlung für Elemente
function startElement($xml_parser, $name, $attr)
```

```
{
...
}
function endElement($xml_parser, $name)
{
...
}
//Ereignisbehandlung für character data
function PCdata($xml_parser, $data)
{
...
}
//Ereignisbehandlung für PIs
function PIdata($xml_parser, $target, $data)
{
...
}
//Parser wird erstellt
$xml_parser = xml_parser_create();

//Ereignisbehandlung für Elemente wird gesetzt
xml_set_element_handler($xml_parser, "startElement",
"endElement");

//Ereignisbehandlung für character data wird gesetzt
xml_set_character_data_handler($xml_parser, "PCdata");

//Ereignisbehandlung für PIs wird gesetzt
xml_set_processing_instruction_handler($xml_parser, "PIdata");

//Ereignisbehandlung für Standardereignisse wird ausgeschaltet
//xml_set_default_handler($xml_parser, "");
?>
```

Listing 15.8: Ein kompletter Parser mit Ereignisbehandlungsroutinen

Das Skript zeigt die komplette Struktur eines Parsers mit allen nötigen Routinen um ein XML-Dokument parsen zu können. Die eigentliche Behandlung der Daten innerhalb der Funktion wurden hier durch Punkte ersetzt, um nicht unnötig Verwirrung zu stiften. Ein komplettes Beispiel mit einer exemplarischen Möglichkeit zur Bearbeitung der Daten finden Sie am Ende dieses Kapitels.

15.5.2 Ein Dokument parsen

Nachdem der Parser komplett aufgesetzt wurde, kann der eigentliche Teil des Programms gestartet werden: das Parsen! Im Gegensatz zur eben geleisteten Vorarbeit geht das Parsen eines XML-Dokuments recht leicht von der Hand, da im Prinzip der Parser nur noch gestartet werden muss. Diese Aufgabe erfüllt die Funktion `xml_parse()`:

```
xml_parse($xml_parser, $str);
```

Die Funktion übernimmt das Handle auf den am Anfang geschaffenen XML-Parser und einen String, der die XML-codierten Daten enthält. Der Parser analysiert den Code Zeichen für Zeichen und löst für die vorher definierten Ereignisse die jeweilige Eventfunktion aus. Die Funktion kann sowohl mit einem kompletten Dokument als auch mit Bruchteilen eines Dokuments arbeiten. Da SAX die Daten nicht aus übergeordnetem Kontext interpretiert, ist das also kein Problem.

Um zu erkennen, ob das Dokument in Teilen fortgesetzt wird, kann ein optionaler Parameter angegeben werden, der automatisch auf TRUE gesetzt wird, wenn das Ende vom Dokument erreicht ist.

```
xml_parse($xml_parser, $str, $end);
```

Die Variable `$end` ist FALSE, solange das letzte Element des XML-Dokuments nicht erreicht wurde. Erst wenn der letzte schließende Tag geparsed wurde, gibt `$end` TRUE zurück.

Die Funktion `xml_parse()` gibt TRUE zurück, wenn das Dokument erfolgreich verarbeitet wurde, ansonsten FALSE. Fehlermeldungen während des Parsens werden in einem der folgenden Kapitel besprochen.

Daten direkt aus einer Datei parsen

Da der XML-Code in der Regel nicht direkt als String im Programm vorliegt, muss die Zieldatei zuerst vom PHP-Skript geöffnet und ausgelesen werden. Die dazu nötigen Funktionen wurden bereits in einem vorhergehenden Kapitel ausführlich besprochen, so dass ich sie als bekannt voraussetze.

Die erste Möglichkeit ist das komplette Dokument auf einmal auszulesen und in einer Variable zu speichern, die dann in einem »Stück« an den Parser übergeben wird. Das Vorgehen ist allerdings bei sehr großen Dokumenten nicht empfehlenswert, da sehr viel Speicher verbraucht wird.

Außerdem würde das alle Vorteile von SAX ad absurdum führen, da genau wie bei DOM das komplette Dokument im Speicher abgebildet wird.

Die zweite und auch elegantere Möglichkeit ist das Parsen des Dokuments »on the fly«, ohne die Daten zwischenzuspeichern. Die Datei wird stückchenweise ausgelesen und im gleichen Zug an den Parser weitergeleitet. Da SAX das teilweise Parsen von Dokumenten unterstützt, ist dieses Vorgehen kein Problem.

```
//Zieldatei wird geöffnet
$fhandle = fopen("data.xml", "r");
while(!feof($fhandle))
{
   $str = fgets($fhandle, 1000);
   //Dokument wird geparsed
   xml_parse($xml_parser, $str);
}
fclose($fhandle);
```

In diesem Beispiel wird die Zieldatei »data.xml« zeilenweise ausgelesen und interpretiert. Realisiert wird dieses Vorgehen durch eine `while()`-Schleife, die als Abbruchbedingung das Ende der Datei mit der Funktion `feof()` bestimmt. Innerhalb der Schleife wird mit `fgets()` eine komplette Zeile ausgelesen und als String an den Compiler übergeben. Wenn die Daten geparsed wurden, schließt sich die Schleife und eine neue Runde beginnt.

Da die Variable `$str` jedes Mal neu überschrieben wird, besteht nicht die Gefahr, dass das komplette XML-Dokument im Speicher abgelegt wird. Die Abbruchbedingung der Schleife könnte selbstverständlich auch über den optionalen Parameter der Funktion `xml_parse()` realisiert werden: Die Schleife bricht ab, sobald das letzte Element erreicht ist. Allerdings werden mit einer solchen Konstruktion fehlerhafte Dokumente nicht berücksichtigt, die eventuell keinen letzten Tag aufweisen. Es kann also zu einer Endlosschleife kommen.

Den Einsatz eines solchen Parsers finden Sie am Ende dieses Kapitels.

Informationen während des Parsens

Während des Parsens ist es möglich, genaue Informationen über den Fortschritt der Analyse zu erhalten. Zu diesem Zweck stellt PHP drei Funktion zur Verfügung, mit denen sich die Position des Parsers bestimmen lässt.

Funktion	Beschreibung
`xml_get_current_line_number()`	Gibt die aktuelle Zeile zurück, wo sich der Parser befindet.
`xml_get_current_column_number()`	Gibt die aktuelle Spalte zurück, wo sich der Parser befindet.
`xml_get_current_byte_index()`	Gibt den aktuellen Byte-Index (Zeichen in der Datei) zurück, wo sich der Parser befindet.

Tabelle 15.7: Informationsfunktionen für den XML-Parser

Alle drei Funktionen erwarten als einzigen Parameter das Handle des Zielparsers, über den die Informationen zurückgegeben werden sollen. Die angeforderten Werte werden als Integer zurückgegeben.

```
$bytes = xml_get_current_byte_index($xml_parser);
$lines = xml_get_current_line_number($xml_parser);
$col = xml_get_current_column_number($xml_parser);

echo "Es wurden schon $lines Reihen und $bytes Byte
geparst. Der Parser befindet sich in Spalte $col.";
```

Ausgabe:

```
Es wurden schon 12 Reihen und 692 Byte geparst. Der Parser befindet
sich in Spalte 0.
```

Wird die Datei zeilenweise ausgegeben, dann wird die Funktion `xml_get_current_column_number()` immer den Wert 0 zurückgeben, weil sie sich immer am Anfang einer neuen Zeile befindet. Eine Möglichkeit, genauere Informationen zu erhalten, wäre das zeichenweise Parsen der Datei.

Einen Parser freigeben

Wenn das XML-Dokument komplett geparsed wurde, wird der XML-Parser nicht mehr benötigt. Um die Systemressourcen zu schonen, besteht die Möglichkeit den reservierten Speicher wieder freizugeben. Diese Aufgabe übernimmt die Funktion `xml_parser_free()`:

```
xml_parser_free($xml_parser);
```

Die Funktion gibt den Speicher wieder frei, der von dem Parser belegt wurde, auf den das übergebene Handle zeigt. Gleichzeitig wird das Handle freigegeben und zeigt nicht mehr auf einen Parser. Die Funktion gibt TRUE zurück, wenn der Speicher erfolgreich freigegeben wurde.

```
$bool=xml_parser_free($xml_parser);
if($bool) {echo "Speicher wieder freigegeben!";}
```

Ausgabe:

```
Speicher wieder freigegeben!
```

Der belegte Speicher wird vom PHP-Interpreter nach Beendigung des Programms automatisch freigegeben.

Den Parser konfigurieren

PHP erlaubt es, einen bestehenden Parser nachträglich zu konfigurieren. Neben der Art der Codierung der Daten, die bereits bei der Deklaration des Parsers festgelegt wird, kann über die Funktion `xml_parser_set_option()` auch die Schreibweise der Tag-Namen beeinflusst werden.

```
xml_parser_set_option($xml_parser, $option, $value);
```

Neben dem Verweis auf den Zielparser erwartet die Funktion zwei weitere Parameter. Der erste Parameter ist eine Konstante, welche die Zieloption im Parser festlegt. Der Wert `$option` kann den Wert einer der folgenden Angaben haben:

Konstante	Option
`XML_OPTION_CASE_FOLDING`	Legt die Schreibweise der Tag-Namen fest (default = TRUE).
`XML_OPTION_TARGET_ENCODING`	Legt die Art der Decodierung fest, mit der der Parser arbeiten soll.

Tabelle 15.8: Konstanten für Parseroptionen

Die erste Konstante beeinflusst die Schreibweise der Übergabewerte der Eventfunktionen für Elemente. Ist `XML_OPTION_CASE_FOLDING` auf TRUE gesetzt, dann werden alle Tag-Namen und Attribute, die als Parameter an die Funktion übergeben werden, komplett in Großbuchstaben geschrieben. Ist der Wert auf FALSE gesetzt, dann werden die Werte so übergeben, wie sie im XML-Dokument auftauchen.

Die zweite Konstante erlaubt es, nachträglich die Art der Codierung, mit der der Parser arbeiten soll, festzulegen. Es werden ISO-8859-1, US-ASCII und UTF-8 unterstützt. Es gelten dieselben Angaben wie bei der

Deklarierung eines Parsers am Anfang des Kapitels. Der Standardwert ist ISO-8859-1.

Als zweiten Parameter erwartet die Funktion `xml_parser_set_option()` den Wert, auf den die zuvor angegebene Zieloption gesetzt werden soll. Je nach Option kann das entweder ein boolscher Wert sein oder ein String mit den Angaben zur Codierung.

```
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, FALSE);
xml_parser_set_option($xml_parser, XML_OPTION_TARGET_ENCODING, "US-
ASCII");
```

Das Gegenstück zu dieser Funktion ist `xml_parser_get_option()`. Über diese Funktion kann der aktuelle Status einer einzelnen Option ausgelesen werden. Die Funktion erwartet zwei Parameter:

```
echo xml_parser_get_option($xml_parser, $option);
```

Neben dem obligatorischen Handle auf den XML-Parser muss nur noch die Zieloption angegeben werden, für die der Status ausgelesen werden soll. Es können dieselben Konstanten verwendet werden wie für die Funktion `xml_parser_set_option()`.

```
$case = xml_parser_get_option($xml_parser, XML_OPTION_CASE_FOLDING);
$enc =  xml_parser_get_option($xml_parser,
XML_OPTION_TARGET_ENCODING);
echo "Case-Folding: $case<br>";
echo "Encoding: $enc";
```

Ausgabe:

```
Case-Folding: 1
Encoding: ISO-8859-1
```

Die Funktion gibt den jeweiligen Wert zurück, der ausgewertet werden kann.

15.5.3 Fehlerbehandlung

Zu der Natur von Regeln und Gesetzen gehört es, dass diese auch gebrochen werden. Egal, ob dies absichtlich oder unabsichtlich geschieht, die Folgen dieses Regelbruchs haben meistens Auswirkungen auf das gesamte Konstrukt der darauf aufbauenden Elemente. Genauso verhält es sich auch bei XML: Sobald Syntax oder der Aufbau eines Dokuments nicht mehr den XML-Konventionen entsprechen, ist es für einen Parser nicht mehr lesbar.

In einem solchen Fall generiert die Funktion `xml_parse()` eine Fehlermeldung, die über die Funktion `xml_get_error_code()` ausgelesen werden kann. Die Funktion ermittelt den Fehlercode der letzten Aktion des Parsers und gibt sie in Form einer Integerzahl zurück. Der Wert 0 entspricht dem Zustand »Kein Fehler aufgetreten«. Jeder weitere Fehler wird durch eine Zahl ungleich 0 beschrieben.

Um die Lesbarkeit der möglichen Fehlerquellen besser darstellen zu können, hat PHP für jeden möglichen Fehlerfall eine Konstante definiert, die den entsprechenden Integerwert speichert:

Konstante	Beschreibung
XML_ERROR_NONE	kein Fehler
XML_ERROR_NO_MEMORY	kein Speicher
XML_ERROR_SYNTAX	Syntaxfehler
XML_ERROR_NO_ELEMENTS	kein Element
XML_ERROR_INVALID_TOKEN	nicht wohlgeformt
XML_ERROR_UNCLOSED_TOKEN	nicht abgeschlossen
XML_ERROR_TAG_MISMATCH	Tag-Fehler
XML_ERROR_DUPLICATE_ATTRIBUTE	doppeltes Attribut
XML_ERROR_ATTRIBUTE_ENTITY_REF	falsche Referenz
XML_ERROR_UNCLOSED_CDATA_SECTION	nicht abgeschlossene Zeichenkette
XML_ERROR_EXTERNAL_ENTITY_HANDLING	falsche Referenz
XML_ERROR_JUNK_AFTER_DOC_ELEMENT	ungültige Zeichen
XML_ERROR_UNDEFINED_ENTITY	nicht definierte Entity
XML_ERROR_BAD_CHAR_REF	falsche Referenz
XML_ERROR_BINARY_ENTITY_REF	falsche Referenz
XML_ERROR_MISPLACES_XML_PI	ungültige PI
XML_ERROR_UNKNOWN_ENCODING	unbekannte Codierung
XML_ERROR_INCORRECT_ENCODING	falsche Codierung

Tabelle 15.9: XML-Fehlercodes

Die Funktion `xml_get_error_code()` gibt den letzten Fehlercode des Parsers zurück. Der einzige Parameter, den die Funktion erwartet, ist das Handle zum Zielparser.

```
$ecode = xml_get_error_code($xml_parser);
```

Über die oben vorgestellten Konstanten ist es nun möglich, bestimmte Fehlermeldungen abzufangen oder generell auf Fehler zu reagieren. In der Regel wird der Parser die Analyse des XML-Dokuments abbrechen und einer Fehlermeldung ausgeben. Um den jeweiligen Fehlercode in eine lesbare Meldung umzuwandeln, muss der Integerwert übersetzt werden.

Über die Funktion `xml_error_string()` ist es möglich, den Fehlercode einem Errorstring zuzuordnen, der im (englischen) Klartext den Fehler beschreibt.

```
$string = xml_error_string($ecode);
```

Die Funktion übernimmt als einzigen Parameter den Errorcode, der von der Funktion `xml_get_error_code()` zurückgegeben wurde. Daraus wird eine entsprechende Fehlermeldung generiert.

Ein typische Fehlerbehandlung innerhalb des Parsers könnte so aussehen:

```
$fhandle = fopen("data.xml", "r");
while(!feof($fhandle))
{
   $str = fgets($fhandle, 1000);
   //Dokument wird geparst
   xml_parse($xml_parser, $str);
   //Die letzte Meldung wird kontrolliert!
   $ecode = xml_get_error_code($xml_parser);
   if($ecode != 0)
   {
      die ("Ein Fehler ist aufgetreten: "
      echo xml_error_string($ecode));
   }
}
fclose($fhandle);
```

Dieses Fragment eines XML-Parsers zeigt, wie eine Fehlerbehandlung in einem Parser integriert werden kann. Das Prinzip ist einfach: Nach jeder Zeile, die vom Parser analysiert wurde, wird die letzte Fehlermeldung abgerufen. Wenn der Errorcode dieser Meldung ungleich 0 ist, liegt ein Fehler vor und das Programm wird mit einer entsprechenden Meldung abgebrochen.

15.5.4 Beispiel für die Verarbeitung von XML-Dokumenten

In diesem Abschnitt werde ich einige Möglichkeiten aufzeigen, wie man mit XML-Dokumenten arbeiten kann. Zu diesem Zweck werde ich zwei verschiedene Parser auf PHP-Basis vorstellen, die dasselbe Dokument zu unterschiedlichen Formaten weiterverarbeiten.

Die Grundlage für diese beiden Programme ist das folgende XML-Dokument, das die Grundzüge einer Datenbank für Personendaten darstellt.

```
<?xml version="1.0"?>
<?coding ISO-8859-1?>
<datenbank>
   <datensatz num="1">
      <name>
         <vorname>Dirk</vorname>
         <nachname>Ammelburger</nachname>
      </name>
      <beruf>Autor</beruf>
      <email>dirk@ammelburger.de</email>
      <web art="privat">http://www.kulturbrand.de</web>
      <web art="geschäftlich">http://www.lastcode.de</web>
   </datensatz>
   <datensatz num="2">
      <name>
         <vorname>Klaus</vorname>
         <nachname>Müller</nachname>
      </name>
      <beruf>Bäcker</beruf>
      <email>klaus@mueller.de</email>
      <web art="privat">http://www.klaus.de</web>
      <web art="geschäftlich">http://www.baecker-klaus.de</web>
   </datensatz>
   <datensatz num="3">
      <name>
         <vorname>Hans</vorname>
         <nachname>Bäcker</nachname>
      </name>
      <beruf>Taxifahrer</beruf>
      <email>hans@baecker.de</email>
      <web art="privat">http://www.hansi.de</web>
    </datensatz>
   <datensatz num="4">
      <name>
         <vorname>Gerda</vorname>
         <nachname>Meier</nachname>
```

```
      </name>
      <beruf>Management</beruf>
      <email>gmeier@schoenundreich.de</email>
      <web art="geschäftlich">http://www.schoenundreich.de</web>
   </datensatz>
   <datensatz num="5">
      <name>
         <vorname>Bill</vorname>
         <nachname>Gates</nachname>
      </name>
      <beruf>Entwickler</beruf>
      <email>BillGates@Microsoft.com</email>
      <web art="privat">http://www.microsoft.com</web>
      <web art="geschäftlich">http://www.billgates.de</web>
   </datensatz>
</datenbank>
```

Listing 15.9: Die Datei data.xml

Die Daten sind in einer Datei mit dem Namen `data.xml` abgespeichert. Beide Applikationen werden auf diese Beispieldaten zugreifen. Die Personendaten spalten sich in fünf verschiedene Informationen auf:

- Vorname
- Nachname
- Beruf
- E-Mail-Adresse
- Webpage

Jeder Wert wird im XML-Dokument durch einen eigenen Tag definiert. Vor- und Nachname der Person werden durch ein weiteres Containerelement mit dem Namen »name« zusammengefasst.

XML zu HTML

Das erste Beispiel zeigt, wie XML-Code durch einen Parser zu HTML-Code umgewandelt wird. Das Ziel ist es, die Daten in einem browsertauglichen Format im Internet darstellen zu können.

```
<?
//Ereignisbehandlung für Elemente
function startElement($xml_parser, $name, $attr)
{
   global $webart;
```

```
   global $num;
   switch ($name)
   {
      case "DATENBANK": echo "<html><center><table width='60%' bor-
der='1'
gcolor='#dddddd'>"; break;
      case "DATENSATZ": $num =  $attr["NUM"]; break;
      case "NAME": echo "<tr><td bgcolor='#bbbbbb'>"; break;
      case "VORNAME": echo "$num. <b>"; break;
      case "NACHNAME": echo " "; break;
      case "BERUF": echo "<tr><td>"; break;
   }
}
function endElement($xml_parser, $name)
{
   switch ($name)
   {
      case "DATENBANK": echo "</table></center></html>"; break;
      case "DATENSATZ": echo "</td></tr>"; break;
      case "NAME": echo "</td></tr>"; break;
      case "NACHNAME": echo "</b>"; break;
      case "BERUF": echo "<br>"; break;
      case "EMAIL": echo "<br>"; break;
      case "WEB": echo "<br>"; break;
   }
}

//Ereignisbehandlung für character data
function PCdata($xml_parser, $data)
{
   echo $data;
}

//Ereignisbehandlung für PIs
function PIdata($xml_parser, $target, $data)
{
   if($target="coding")
   {
     xml_parser_set_option($xml_parser, XML_OPTION_TARGET_ENCODING,
data);
   }
}

//Parser wird erstellt
$xml_parser = xml_parser_create();

//Ereignisbehandlung für Elemente wird gesetzt
```

```
xml_set_element_handler($xml_parser, "startElement", "endElement");
//Ereignisbehandlung für character data wird gesetzt
xml_set_character_data_handler($xml_parser, "PCdata");

//Ereignisbehandlung für PIs wird gesetzt
xml_set_processing_instruction_handler($xml_parser, "PIdata");

//Ereignisbehandlung für Standardereignisse wird ausgeschaltet
//xml_set_default_handler($xml_parser, "");


//Zieldatei wird geöffnet
$fhandle = fopen("data.xml", "r");
while(!feof($fhandle))
{
   $str = fgets($fhandle, 1000);

   //Dokument wird geparsed
   xml_parse($xml_parser, $str);

   //Fehlerbehandlung
   $ecode = xml_get_error_code($xml_parser);
   if($ecode != 0) {
   die ("<script>alert('Ein Fehler ist aufgetreten:“
.xml_error_string($ecode)."');</script>");}
}
fclose($fhandle);
echo "<br>Dokumentgröße: ".xml_get_current_byte_index($xml_parser)."
Byte";
//Parser wird freigegeben
xml_parser_free($xml_parser);
?>
```

Listing 15.10: XML zu HTML

Das Skript bietet vom Funktionsumfang und vom Aufbau her nicht viel Neues. Wenn alles geklappt hat, sollte folgendes Bild im Browser erscheinen (siehe Abbildung 15.5).

Die Daten werden in einer Tabelle übersichtlich präsentiert, ohne sich dabei von der vorgegebenen XML-Struktur gelöst zu haben. Den Aufbau des HTML-Dokuments wird über die Eventhandler gesteuert, die je nach Element den geeigneten HTML-Baustein ausgeben.

1. **Dirk Ammelburger**
Autor dirk@ammelburger.de http://www.kulturbrand.de http://www.lastcode.de
2. **Klaus Müller**
Bäcker klaus@mueller.de http://www.klaus.de http://www.baecker-klaus.de
3. **Hans Bäcker**
Taxifahrer hans@baecker.de http://www.hansi.de
4. **Gerda Meier**
Management gmeier@schoenundreich.de http://www.schoenundreich.de
5. **Bill Gates**
Entwickler BillGates@Microsoft.com http://www.microsoft.com http://www.billgates.de

Abbildung 15.5: Darstellung von XML-Daten im HTML-Format

Über die PI-Anweisung »coding« kann während des Parsens die Codierung des Parsers angepasst werden. Sollte ein XML-Dokument mehrere Zeichensätze verwenden (was nicht empfehlenswert ist), können diese so unterstützt werden.

Eventuelle Fehler im XML-Dokument werden über einen Abfangmechanismus während des Parsens aufgedeckt und angezeigt. Dazu wird eine JavaScript-AlertBox verwendet, die vor der Seite aufpoppt und den Fehler anzeigt:

Abbildung 15.6: Fehlerwarnung beim Parsen

Das Programm wird im Fehlerfall automatisch abgebrochen, da die korrekte Darstellung der Daten nicht mehr gewährleistet werden kann.

XML zu Excel

Das zweite Beispiel zeigt, wie derselbe XML-Code aus dem letzten Beispiel in eine Exceltabelle umgewandelt wird. Da der ganze Vorgang online auf einem entfernten Server passiert und wir keinen direkten Zugriff auf die Excelsoftware haben, verwenden wir einen kleinen Trick, der es uns erlaubt, relativ einfach Exceldokumente zu erzeugen.

Die Tabellenkalkulation Microsoft Excel verwendet für die Speicherung der erstellten Dokumente normalerweise ein eigenes propritäres Format, das nicht ohne weiteres von einer Applikation kopiert werden kann. Um trotz allem kompatibel zu anderen Textverarbeitungsprogrammen oder einfachen Editoren zu bleiben unterstützt Excel so genannte *tab-limited documents*, die alle Daten einer Tabelle in eine Textdatei speichern und durch Tabulator-Zeichen trennen.

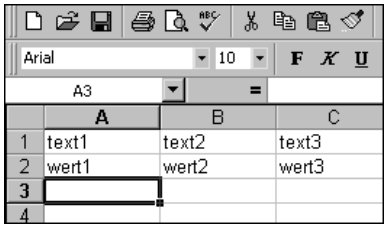


Abbildung 15.7: Exceltabelle

Eine Exceldatei, die diesen Aufbau hat, kann durch die Verwendung der Sonderzeichen `\n` (Zeilenumbruch) und `\t` (Tabulator) so dargestellt werden:

```
text1\ttext2\ttext3\nwert1\twert2\twert3\n
```

In Excel werden die Daten automatisch konvertiert und korrekt dargestellt. Auf diese Weise ist es möglich, dynamische Tabellen für die bekannte Tabellenkalkulation »on-the-fly« auf einer Webseite zu erstellen.

Damit der Browser auch versteht, welches Format wir über das Netz schicken, und es auch richtig darstellt, muss am Anfang des Scripts der Excel-Header angegeben werden. Dazu benutzen Sie die schon bekannte Funktion `header()`.

```
header("Content-Type: application/vnd.ms-excel");
```

Mit den Voraussetzungen können Sie nun ohne weiteres Exceltabellen – auch ohne XML – erstellen.

Das Beispiel zeigt jetzt, wie diese Technik in der Praxis zur Darstellung des XML-Dokuments in einer Exceltabelle genutzt werden kann. Noch einmal zur Erinnerung: Es wird dasselbe XML-Dokument verwendet, das bereits im letzten Beispiel in HTML konvertiert wurde.

```
<?
//Excelheader wird generiert
header("Content-Type: application/vnd.ms-excel");
//Überschriften werden ausgegeben
echo "Nummer\tVorname\tNachname\tBeruf\teMail\tPrivat\tGeschäft\n";
//Ereignisbehandlung für Elemente
function startElement($xml_parser, $name, $attr)
{
   global $num;
   switch ($name)
   {
      case "DATENSATZ": $num =  $attr["NUM"]; echo "\n";  break;
      case "VORNAME": echo "$num.\t"; break;
      case "WEB": if ($attr["ART"] == "geschäftlich") {echo "\t";}
break;
   }
}
function endElement($xml_parser, $name)
{
   switch ($name)
   {
      case "VORNAME": echo "\t"; break;
      case "NACHNAME": echo "\t"; break;
      case "BERUF": echo "\t"; break;
      case "EMAIL": echo "\t"; break;
   }
}
//Ereignisbehandlung für character data
function PCdata($xml_parser, $data)
{
   echo $data;
}
//Ereignisbehandlung für PIs
function PIdata($xml_parser, $target, $data)
{
   if($target="coding")
```

```
   {
      xml_parser_set_option($xml_parser, XML_OPTION_TARGET_ENCODING,
data);
   }
}
//Parser wird erstellt
$xml_parser = xml_parser_create();
//Ereignisbehandlung für Elemente wird gesetzt
xml_set_element_handler($xml_parser, "startElement", "endElement");
//Ereignisbehandlung für character data wird gesetzt
xml_set_character_data_handler($xml_parser, "PCdata");
//Ereignisbehandlung für PIs wird gesetzt
xml_set_processing_instruction_handler($xml_parser, "PIdata");
//Ereignisbehandlung für Standardereignisse wird ausgeschaltet
//xml_set_default_handler($xml_parser, "");
//Zieldatei wird geöffnet
$fhandle = fopen("data.xml", "r");
while(!feof($fhandle))
{
   $str = fgets($fhandle, 1000);
   //Dokument wird geparst
   xml_parse($xml_parser, $str);
   //Fehlerbehandlung
   $ecode = xml_get_error_code($xml_parser);
   if($ecode != 0) {
   die ("Ein Fehler ist aufgetreten: ".xml_error_string($ecode));}

}
fclose($fhandle);
//Parser wird freigegeben
xml_parser_free($xml_parser);
?>
```

Listing 15.11: XML wird zu einer Exceltabelle umgewandelt

Das Skript ist dem ersten Beispielskript sehr ähnlich: Die Unterschiede liegen eigentlich nur in der Formatierung der Daten, die jetzt für ein Excel-Sheet ausgerichtet werden müssen. Anstelle der HTML-Formatierungsanweisungen werden jetzt Sonderzeichen wie `\t` und `\n` verwendet.

Je nach Browsereinstellungen wird beim Aufruf des Skripts entweder sofort ein Excelfenster geöffnet, oder der Broser fragt nach, wie mit den Daten verfahren werden soll. In der Regel öffnet sich ein Abfragefenster, das in etwa so aussieht:

Abbildung 15.8: Browserhinweis

Hier können Sie wählen, ob die Daten auf der Festplatte gespeichert (»Datei auf Datenträger speichern«) oder sofort im Browser angezeigt werden sollen (»Die Datei von ihrem aktuellen Ort öffnen«).

Wenn Sie die Datei speichern, erhalten Sie automatisch eine Exceldatei, die Sie per Doppelklick öffnen können. Öffnen Sie die Datei direkt, wird ein Excelfenster im Browser erzeugt, das die Daten sofort anzeigt. Egal welchen Weg Sie gehen, das Ergebnis sieht etwa so aus:

G1 = Geschäft

	A	B	C	D	E	F	G	H
1	Nummer	Vorname	Nachname	Beruf	eMail	Privat	Geschäft	
2								
3	1.	Dirk	Ammelburger	Autor	dirk@ammelburger.de	http://www.kulturbrand.de	http://www.lastcode.de	
4	2.	Klaus	Müller	Bäcker	klaus@mueller.de	http://www.klaus.de	http://www.baecker-klaus.de	
5	3.	Hans	Bäcker	Taxifahrer	hans@baecker.de	http://www.hansi.de		
6	4.	Gerda	Meier	Management	gmeier@schoenundreich.de		http://www.schoenundreich.de	
7	5.	Bill	Gates	Entwickler	BillGates@Microsoft.com	http://www.microsoft.com	http://www.billgates.de	
8								

Abbildung 15.9: XML-Daten als Exceltabelle

15.5.5 Übungen

Ü

1. Was ist der Unterschied zwischen XML und HTML?
2. Welchen Vorteil bietet XML?
3. Was ist der Unterschied zwischen einem Element und einem Attribut?
4. Was sind PIs?

5. Warum wird der folgende XML-Code von jedem Parser ignoriert?

```
<!-- <test>Das ist ein Test!/<test> -->
```

6. Welche Aufgabe hat eine DTD?
7. Was unterscheidet konkrete Daten von abstrakten Einheiten?
8. Erklären Sie den Unterschied zwischen wohlgeformten und gültigen Dokumenten.
9. Wann sollte eine externe DTD einer internen vorgezogen werden?
10. Erklären Sie das SAX-Modell.

15.6 Das DOM-Model

Das DOM-Model ist neben dem eventgestützten Parsen von Daten eine weitere Möglichkeit XML-Dokumente zu bearbeiten. DOM steht für *Document Object Model* und bezeichnet ein Verfahren, ein XML-Dokument nicht als reine Aufreihung von Tags und Informationen zu sehen, sondern die kompletten Daten als Baum zu repräsentieren.

DOM liest das gesamte XML-Dokument in den Speicher ein und stellt so alle Daten dem Programmierer zur Verfügung. Der Zugriff auf die Daten erfolgt über so genannte Knoten, die jeweils ein Datenstück aus dem Dokument repräsentieren. Das DOM-Modell hat gegenüber dem eventgestützten Verfahren von PHP den Vorteil, dass man Dokumente nicht nur parsen, sondern auch manipulieren kann. Das DOM-Modul für PHP erlaubt es, sogar komplette XML-Dokumente neu anzulegen.

Bei aller Freude über DOM darf man allerdings nicht vergessen, dass dieses Model auch einen entscheidenen Nachteil hat: Da DOM das gesamte Dokument in den Speicher liest, können die zur Verfügung stehenden Ressourcen auf dem Server stark in Anspruch genommen werden, was die Verarbeitung großer Dokumente verlangsamt. Je größer und komplexer das XML-Dokument ist, desto deutlicher wird der Performanceverlust.

Bevor Sie also entscheiden, welche der beiden Varianten Sie für Ihr Projekt verwenden wollen, müssen Sie festlegen, ob Sie Dokumente nur parsen oder auch erstellen wollen. Zur Erstellung einfacher Dokumente ist die Verwendung des DOM-Moduls unter Umständen nicht nötig, da das Schreiben von Textdateien auch so einfach zu realisieren ist.

Für den Einsatz des DOMXML-Moduls unter PHP 4 benötigen Sie die optionale DOM-Bibliothek. Diese Software ist nicht in der Standarddistribution der aktuellen PHP-Version vorhanden. Fragen Sie Ihren Systemadministrator, ob das DOM-Modul auf ihrem Server installiert ist oder ob er bereit ist dieses Paket nachträglich zu installieren.

Möchten Sie das Modul lokal nutzen, dann müssen Sie es unter folgender Adresse herunterladen:

http://www.xmlsoft.org/

Kopieren Sie die Datei `php_domxml.dll` in das Systemverzeichnis Ihres Windowsbetriebsystems und aktualisieren Sie die Konfigurationsdatei php.ini. Die Datei muss die folgenden zwei Einträge beinhalten:

```
extension_dir=c:\windows\system
extension=php_domxml.dll
```

Das Verzeichnis, in dem der PHP-Interpreter nach den Erweiterungsmodulen suchen soll, muss an den Systempfad Ihres Rechners angepasst werden. Unter Windows NT oder Windows 2000 heißt der Systempfad standardmäßig

```
c:\winnt\system32
```

Nachdem Sie alle nötigen Einstellungen vorgenommen haben, muss der Apache Server neu gestartet werden. Danach sollten Ihnen alle Features der DOMXML-Bibliothek zu Verfügung stehen.

15.6.1 Ein XML-Objekt erzeugen

Die Grundlage für den Umgang mit den DOM-XML-Funktionen ist ein Klassenobjekt, das den zu bearbeitenden XML-Code repräsentiert. Dieses Objekt ist nicht vergleichbar mit dem Handle zum XML-Parser, der im letzten Kapitel eingesetzt wurde. Dieses Datenobjekt steht nicht für einen Parser, sondern für die Daten, die analysiert werden sollen.

Die Funktion `xmldoc()` erschafft ein XML-Objekt aus einem String, der den zu analysierenden XML-Code enthält. Die Anwendung ist recht einfach:

```
$doc = xmldoc($dokument);
```

Der einzige Parameter ist eine Stringvariable, die den XML-Code enthält. Die Funktion gibt bei Erfolg ein Objekt zurück, das das übergebene XML-Dokument repräsentiert. Das erschaffene Objekt ist Voraussetzung für alle weiteren Schritte.

Da der XML-Code im seltensten Falle schon beim Programmstart als Variable im Speicher vorliegt, ist es meistens nötig, die Daten aus einer Datei auszulesen. Für alle folgenden Beispiele verwende ich wieder den bekannten XML-Code aus der Datei »data.xml«, den Sie bereits im letzten Kapitel kennen gelernt haben.

Im Gegensatz zum SAX-Parser, der die Datei schrittweise ausgelesen und interpretiert hat, ist es bei Document Object Model nicht nötig, die Datei zeilenweise zu übergeben. Ganz im Gegenteil: Der Klassenkonstruktor der Funktion `xmldoc()` verlangt, dass der XML-Code in »einem Stück« übergeben wird. Der einfachste Weg diese Bedingung zu erfüllen, ist die Datei komplett auszulesen, bevor die Daten übergeben werden.

Es gibt verschiedene Möglichkeiten diese Vorgaben in die Tat umzusetzen. Meiner Meinung nach ist eine Kombination aus den Funktionen `implode()` und `file()` die einfachste Möglichkeit.

```
$dokument = implode("", file("data.xml"));
```

Die Funktion `file()` liest die Zieldatei komplett aus und gibt sie als zeilenweises Array zurück. Um die Daten zu einem kompletten String zusammenzufügen, verwenden Sie die Funktion `implode()`, die die Elemente ohne Trennzeichen zusammensetzt. Vielleicht nicht umwerfend elegant, aber sicherlich effektiv. Das Ergebnis ist ein String, der den kompletten Inhalt der Datei speichert.

```
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
```

Mit zwei einfachen Zeilen Code ist der erste Schritt also getan. Es steht jetzt ein XML-Objekt zur Verfügung, das auf den gesamten Inhalt der Datei im Speicher zeigt.

15.6.2 Das XML-Objekt

Das neu erzeugte XML-Objekt gibt Ihnen jetzt die Möglichkeit mit den übergebenen Daten zu arbeiten. Vorweg möchte ich aber ein paar Worte zur Praxis dieses Moduls verlieren, da die Anwendung ein wenig schizo-

phren anmuten kann. Die Funktionalität des DOM-Moduls ist zum Teil mit objektorientierter Syntax realisiert worden und zum Teil mit »normalen« Funktionen. Teilweise sind die Anwendungsmöglichkeiten auch doppelt vorhanden, so dass man die Qual der Wahl hat.

Im ersten Schritt möchte ich die Möglichkeiten des XML-Objekts erklären, das durch die Funktion `xmldoc()` erschaffen wurde. Das Objekt verfügt über eine Reihe von Methoden und Eigenschaften, die es ohne Probleme erlauben, Informationen über das XML-File abzurufen.

Methode / Eigenschaft	Beschreibung
`root()`	gibt ein Objekt zum Wurzelelement des Dokuments zurück
`dumpmem()`	gibt die Daten des Dokuments zurück
`version`	Version des XML-Dokuments
`encoding`	Codierung des XML-Dokuments
`standalone`	gibt TRUE zurück, wenn das Dokument keine externen Verweise (DTD) hat

Tabelle 15.10: Methoden und Eingenschaften des XML-Objekts

Die drei Objekteigenschaften `version`, `encoding` und `standalone` sind einfache Möglichkeiten allgemeine Informationen über das Objekt auszulesen. Die Daten werden über die bereits bekannte objektorientierte Syntax abgerufen.

```
$doc->encoding;
$doc->version;
$doc->standalone;
```

Die Eigenschaft `encoding` gibt die Bezeichnung des Zeichensatzes zurück, der in der PI *<?xml …?>* über den Parameter encoding angegeben wurde. Die Eigenschaft ist nur gesetzt, wenn diese Information auch tatsächlich im XML-Dokument angegeben wurde. Dasselbe gilt für die Eigenschaft `version`, die die XML-Version des Dokuments speichert. Auch hier wird die Information über die Processing Instruction *<?xml …?>* übergeben.

Die dritte Eigenschaft `standalone` ist ein boolescher Wert, der die Situation des XML-Dokuments in einem Netzwerk beschreibt. Verweist das Dokument innerhalb des Codes auf eine weitere Datei, wie beispielsweise

eine externe DTD, ist dieser Wert FALSE. Ist das Dokument unabhängig von weiteren Dateien und hat keine Verweise nach außen, dann gibt standalone TRUE zurück.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
echo $doc->encoding;
echo "<br>";
echo $doc->version;
echo "<br>";
echo $doc->standalone;
?>
```

Listing 15.12: Eigenschaften des XML-Objekts

Ausgabe:

```
ISO-8859-1
1.0
-1
```

Die Funktion `dumpmem()` gehört zu den interessantesten Methoden des XML-Objekts. Sie erlaubt es, das komplette gespeicherte XML-Dokument auszugeben und nach eventuellen Veränderungen wieder in einer Datei zu speichern.

```
<pre>
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
echo htmlentities($doc->dumpmem());
?>
</pre>
```

Listing 15.13: Die Funktion dumpmem()

Ausgabe:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<datenbank>
   <datensatz num="1">
      <name>
         <vorname>Dirk</vorname>
         <nachname>Ammelburger</nachname>
      </name>
      <beruf>Autor</beruf>
      <email>dirk@ammelburger.de</email>
```

```
      <web art="privat">http://www.kulturbrand.de</web>
      <web
art="geschäftlich">http://www.lastcode.de</web>
   </datensatz>
   <datensatz num="2">
      <name>
         <vorname>Klaus</vorname>
         <nachname>Müller</nachname>
      </name>
      <beruf>Bäcker</beruf>
      <email>klaus@mueller.de</email>
      <web art="privat">http://www.klaus.de</web>
      <web art="geschäftlich">http://www.baecker-
klaus.de</web>
   </datensatz>
   <datensatz num="3">
      <name>
         <vorname>Hans</vorname>
         <nachname>Bäcker</nachname>
      </name>
      <beruf>Taxifahrer</beruf>
      <email>hans@baecker.de</email>
      <web art="privat">http://www.hansi.de</web>
    </datensatz>
   <datensatz num="4">
      <name>
         <vorname>Gerda</vorname>
         <nachname>Meier</nachname>
      </name>
      <beruf>Management</beruf>
      <email>gmeier@schoenundreich.de</email>
      <web
art="geschäftlich">http://www.schoenundreich.de</web>
   </datensatz>
   <datensatz num="5">
      <name>
         <vorname>Bill</vorname>
         <nachname>Gates</nachname>
      </name>
      <beruf>Entwickler</beruf>
      <email>BillGates@Microsoft.com</email>
      <web art="privat">http://www.microsoft.com</web>
      <web art="geschäftlich">http://www.billgates.de</web>
   </datensatz>
</datenbank>
```

Da die Rückgabe der Daten in korrekter XML-Codierung erfolgt, muss die Ausgabe von `dumpmem()` für die Darststellung im Browser mit der Funktion `htmlentities()` überarbeitet werden. Die Funktion codiert alle spitzen Klammern im String mit den Kürzeln `<` und `>` und erlaubt es so, die Daten im Browser auszugeben.

Die eigentliche Aufgabe von `dumpmem()` liegt allerdings nicht in der Darstellung der Daten, sondern vielmehr in der Möglichkeit das bearbeitete XML-Dokument wieder in einem String zu speichern. So ist es problemlos möglich, einen durch DOM-Funktionen bearbeiteten oder erstellten XML-Baum wieder bequem in einer Datei zu speichern. Die Methode wird also in einem späteren Kapitel eine große Rolle spielen.

15.6.3 Ein XML-Dokument mit DOM parsen

Die für den jetzigen Zeitpunkt mit Abstand wichtigste Methode des XML-Objekts ist `root()`, die ein Objekt auf das Root-Element des XML-Dokumentes zurückgibt. Dieses Element ist der Ausgangspunkt für alle weiteren Zugriffe auf das Dokument, egal ob Daten gelesen oder geschrieben werden sollen. Die Methode wird ohne Parameter aufgerufen:

```
$root=$doc->root();
```

Das zurückgegebene Objekt beschreibt das Root-Element mit allen folgenden Elementen durch eine Reihe von Methoden und Eigenschaften. Dasselbe Objekt lässt sich auch über die Funktion `domxml_root()` erschaffen, die genau denselben Effekt hat wie die `root()`-Methode.

```
$root=domxml_root($doc);
```

Der Unterschied liegt schlicht in der Syntax, so das die Technik frei nach Geschmack eingesetzt werden kann. Egal wie Sie das neue Objekt erschaffen, es weist immer folgende Methoden und Eigenschaften auf.

Methode/Eigenschaft	Beschreibung
`children()`	gibt ein Array mit Objekten auf alle untergeordneten Elemente zurück
`type`	gibt den Typ des Elements zurück

Tabelle 15.11: Eigenschaften und Methoden des Root-Elements

Methode/Eigenschaft	Beschreibung
`getattr()`	gibt den Wert eines bestimmten Attributs zurück
`attributes()`	gibt ein Array mit allen Attributen des Elements zurück
`name`	gibt den Namen des Elements zurück
`content`	gibt den Inhalt des Elements zurück

Tabelle 15.11: Eigenschaften und Methoden des Root-Elements (Forts.)

Im *Document Object Model* werden alle Datenobjekte eines XML-Dokuments als Knoten bezeichnet. Dabei spielt es keine Rolle, ob das Element in weitere Unterelemente verzweigt wird oder ob der Tag nur Daten enthält. Die Bezeichnung Knoten ist aus der Vorstellung entstanden, dass in einem XML-Datenbaum alle Information verknüpft sind und über die Schnittpunkte an den Elementen »verknotet« werden.

Die grundlegenden Eigenschaften eines Knotens können immer über die Eigeschaften des Knotenobjekts ausgelesen werden. Da das Root-Element ebenfalls ein Knoten ist, verfügt es über die Eigenschaften `name` und `content`, wie es für alle Knoten der Fall ist.

Die Eigenschaft `name` gibt die Bezeichnung des Elements zurück, die das aktuelle Knotenobjekt symbolisiert. Wird die Eigenschaft für das Root-Element ausgeführt, wird der Name des ersten Elements des XML-Dokuments ausgegeben. Ähnlich verhält es sich mit der Eigenschaft `content`, die den Inhalt eines Elements zurückgibt. Allerdings bezieht sich diese Eigenschaft nicht nur auf das aktuelle Element, sondern auf folgenden Unterelemente des Knotens. Wird dieser Punkt in einem hochpositionierten Knoten wie dem Root-Element ausgeführt, kann der Rückgabestring wenig strukturiert sein.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root=$doc->root();
echo $root->name;
echo "<br>";
echo $root->content;
?>
```

Listing 15.14: Die Eigenschaften name und content

Ausgabe:

```
datenbank

Dirk Ammelburger Autor dirk@ammelburger.de http://www.kulturbrand.de
http://www.lastcode.de Klaus Müller Bäcker klaus@mueller.de http://
www.klaus.de http://www.baecker-klaus.de Hans Bäcker Taxifahrer
hans@baecker.de http://www.hansi.de Gerda Meier Management
gmeier@schoenundreich.de http://www.schoenundreich.de Bill Gates
Entwickler BillGates@Microsoft.com http://www.microsoft.com http://
www.billgates.de
```

In einem solchen Fall empfiehlt es sich, das Dokument detaillierter zu parsen, um die Informationen genauer auslesen zu können.

Den Typ des Knotens feststellen

Die Eigenschaft type gibt den Typus des Elements zurück, auf den das aktuelle XML-Dokument zeigt. Die Rückgabe dieser Eigenschaft ist ein Integerwert zwischen 1 und 12, der jeweils für einen bestimmten Typ von Knoten steht. Jeder dieser Werte ist durch eine XML-Konstante definiert, die die Identifikation der Knotenpunkte erleichtert.

Konstante	Wert	Beschreibung
XML_ATTRIBUTE_NODE	1	Element
XML_ELEMENT_NODE	2	Attribut
XML_TEXT_NODE	3	Text
XML_CDATA_SECTION_NODE	4	Daten
XML_ENTITY_REF_NODE	5	Entity-Referenz
XML_ENTITY_NODE	6	Entity-Element
XML_PI_NODE	7	PI-Element
XML_COMMENT_NODE	8	Kommentar
XML_DOCUMENT_NODE	9	Dokument (Root-Element)
XML_DOCUMENT_TYPE_NODE	10	Dokumententyp
XML_DOCUMENT_FRAG_NODE	11	Fragment
XML_NOTATION_NODE	12	Information

Tabelle 15.12: XML-Konstanten

Jedes XML-Objekt kann über die `type`-Eigenschaft identifiziert werden. Es gibt keine Funktion, die diese Information ohne OOP-Syntax zur Verfügung stellt. Das folgende Beispiel zeigt, wie `type` in der Praxis eingesetzt wird.

```
<?
function getNodeType($typ)
{
   switch ($typ)
   {
      case XML_ELEMENT_NODE: return "Element"; break;
      case XML_ATTRIBUTE_NODE: return "Attribut"; break;
      case XML_TEXT_NODE: return "Text"; break;
      case XML_CDATA_SECTION_NODE: return "Daten"; break;
      case XML_ENTITY_REF_NODE: return "Entity"; break;
      case XML_ENTITY_NODE: return "Entity-Knoten"; break;
      case XML_PI_NODE: return "PI-Knoten"; break;
      case XML_COMMENT_NODE: return "Kommentar"; break;
      case XML_DOCUMENT_NODE: return "Dokument"; break;
      case XML_DOCUMENT_TYPE_NODE: return "Dokumententyp"; break;
      case XML_DOCUMENT_FRAG_NODE: return "Fragment"; break;
      case XML_NOTATION_NODE: return "Information"; break;
      default: return "unbekannt";
   }
}
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$typ = $doc->type;
echo "Das Dokument \$doc ist vom Typ ". getNodeType($typ);
$root=$doc->root();
$typ = $root->type;
echo "<br>Das Element ". $root->name . " ist vom Typ: " . getNode-
Type($typ);
?>
```

Listing 15.15: Die Eigenschaft type

Ausgabe:

```
Das Dokument $doc ist vom Typ Dokument
Das Element datenbank ist vom Typ: Element
```

Das Skript ist relativ einfach und demonstriert, wie mittels der Eigenschaft `type` und den dazugehörigen XML-Konstanten der jeweilige Typ des Knotens bestimmt werden kann. In der Praxis ist es meistens nicht erforderlich, die Typbezeichnung des Elements in einen String umzuwandeln und auszugeben.

Attribute eines Knotens bestimmen

XML erlaubt es, die Baumstruktur des XML-Dokuments durch Attribute mit weiteren Informationen zu versehen. Die Daten werden innerhalb des Tag-Elements durch Name-Wert-Paare übergeben und können durch verschiedene Methoden im *Document Object Model* ausgelesen werden.

Die Methode `attributes()` steht für alle Knotenobjekte sowie dem Root-Objekt eines XML-Baumes zur Verfügung. Über diese Methode ist es möglich, die Attribute eines Elements auszulesen und sowohl die Bezeichnung als auch den Wert zurückzugeben. Der Zugriff erfolgt über eine Reihe von Attributobjekten, die in Form eines Arrays von der Methode zurückgegeben werden.

```
$attr = $root->attributes();
```

Je nach Anzahl der Attribute eines Elements hat das zurückgegebene Array unterschiedlich viele Elemente. Besitzt der Knoten keine Attribute, dann ist das Array leer. Das nicht objektorientierte Gegenstück der Methode `attributes()` ist die Funktion `domxml_attributes()`, die denselben Effekt hat. Die Funktion bekommt ein Knotenobjekt als Parameter übergeben und gibt das Array mit den Attributobjekten des Elements zurück.

```
$attr = domxml_attributes($root);
```

Dem neu erschaffenen Attributobjekt stehen folgende Möglichkeiten zur Verfügung:

Eigenschaft	Beschreibung
`name`	Gibt den Namen des Attributs zurück
`content`	Gibt den Wert des Attributs zurück

Tabelle 15.13: Das Attributobjekt

Der Zugriff auf die einzelnen Attributobjekte erfolgt über ein nummerisches Array, das pro Element ein Objekt speichert. Wir nehmen für die folgenden Beispiele an, dass das Root-Element des XML-Dokuments über folgenden Aufbau verfügt:

```
<datenbank name="Personendaten" datensaetze="5">
```

Das Rückgabearray mit den Attributobjekten hat also zwei Elemente, die über Nummern 0 und 1 abgesprochen werden können. Das folgende Skript zeigt, wie die Werte über das jeweilige Objekt ausgelesen werden können.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root = domxml_root($doc);

$attr = domxml_attributes($root);

echo $attr[0]->name;
echo ": ";
echo $attr[0]->value;
echo "<br>";
echo $attr[1]->name;
echo ": ";
echo $attr[1]->value;
?>
```

Listing 15.16: Das Attributobjekt

Ausgabe:

```
name: Personendaten
datensaetze: 5
```

Die Auswertung von Attributen gestaltet sich also relativ einfach. Wenn man bereits weiß, wie die Attribute eines Elements heißen, dann ist die Analyse eines Knotens noch einfacher. Dank der Methode `getattr()`, die jedem Knotenobjekt zur Verfügung steht, ist es gar nicht nötig, ein Attributobjekt zu erstellen. Der jeweilige Wert kann direkt über seinen spezifischen Namen ausgelesen werden.

```
$root->getattr("name");
```

Die Attributsbezeichnung wird der Methode als Parameter übergeben, die über das jeweilige Knotenobjekt aufgerufen wird. Im Beispiel handelt es sich um das aktuelle Root-Objekt. Dieselbe Aufgabe wie die `getattr()`-Methode wird auch von der DOM-XML-Funktion `domxml_getattr()` erledigt. Die Funktion erwartet zwei Parameter und gibt ebenfalls den Wert eines Attributs zurück.

```
domxml_getattr($root, "name");
```

Der erste Parameter bezeichnet den Knotenpunkt, der das Zielattribut enthält, und der zweite Parameter übergibt den Namen des Attributs. In der Praxis besteht kein Unterschied zwischen `getattr()` und `domxml_getattr()`.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root=domxml_root($doc);
echo $root->getattr("name");
echo "<br>";
echo domxml_getattr($root, "datensaetze");
?>
```

Listing 15.17: Die Methode getattr() und die Funktion domxml_getattr()

Ausgabe:

```
Personendaten
5
```

Sowohl die Funktion `domxml_getattr()` als auch die Methode `getattr()` geben den Wert des jeweiligen Attributs aus. In der Praxis werden diese Möglichkeiten der Methode `attributes()` meistens vorgezogen, da der Aufbau eines Dokuments durch die DTD bekannt ist. Außerdem entfällt die Erschaffung eines weiteren Objekts im Programm.

Im XML-Dokument navigieren

Nachdem die grundsätzliche Informationstruktur eines einzelnen Knotens am Beispiel des Root-Elements erläutert wurde, können wir uns nun um die restlichen Daten des XML-Dokuments kümmern.

Jedes Element, das weitere Unterelemente umschließt (also ein Containerelement), verfügt über die Methode `children()`, die ein Array mit Objekten generiert. Jedes Element in der Liste ist ein Objekt, welches eines der Unterelemente des aktuellen Knotens symbolisiert.

```
$node = $root->children();
```

Die Methode erschafft sozusagen eine Zugriffsmöglichkeit auf die Kinder des aktuellen Knotens, über den die Methode `children()` aufgerufen wird. Jeder Knoten im Array ist wieder ein vollständiges Knotenobjekt, das über alle Methoden und Eigenschaften verfügt, die im ersten Teil dieses Kapitels besprochen wurden.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root=domxml_root($doc);

$node = $root->children();
echo $node[0]->name;
echo "<br>";
echo $node[0]->content;
echo "<br>";
echo $node[0]->type;
echo "<br>";
echo $node[0]->getattr("num");
?>
```

Listing 15.18: Ein neues Knotenelement

Ausgabe:

```
datensatz
DirkAmmelburgerAutordirk@ammelburger.dehttp://www.kultur-
brand.dehttp://www.lastcode.de
1
1
```

Das erste Element des Arrays speichert ein Knotenobjekt, das das erste »Datensatz«-Element im XML-Dokument symbolisiert. Die Informationen werden entsprechend dieser Tatsachen ausgegeben.

Die Eigenschaft `name` gibt den Namen `datensatz` zurück, da das Element so bezeichnet wird. Über die Eigenschaft `content` wird der gesamte Textinhalt des Elements sowie aller Unterlemente ausgegeben. Da es sich hierbei nur um die Daten des ersten Datensatzes handelt, ist die Informationsflut natürlich deutlich geringer als beim Root-Element. Für den Typ wird der Integerwert 1 zurückgegeben, der für ein Element steht. Der letzte Methodenaufruf liest das Attribut `num` des Elements aus, das logischerweise auch den Wert 1 speichert.

Dasselbe Ergebnis wie die Methode `children()` eines Knotenobjekts liefert auch die PHP-Funktion `domxml_children()`.

```
$node=domxml_children($root);
```

Sie bekommt als Parameter ein Knotenobjekt übergeben und liefert ein Array mit den Unterelementen des Knotens zurück. Technisch gesehen besteht kein Unterschied zwischen `children()` und `domxml_children()`.

Zusätzlich zu den oben bereits besprochenen Methoden und Eigenschaften vom Root-Knoten verfügt jeder weitere Childknoten über folgende Möglichkeiten.

Methoden	Beschreibung
`parent()`	gibt ein Objekt mit dem übergeordneten Element des Knotens zurück
`lastchild()`	gibt das letzte untergeordnete Objekt zurück (auch für das Root-Element)

Tabelle 15.14: Weitere Knotenmethoden

Die Methode `parent()` ist das Gegenstück zur Methode `children()` da sie immer das übergeordnete Element eines Knotens zurückgeben.

```
$parent_node = $node[1]->parent();
```

Dabei ist es selbstverständlich egal, um welches Unterelement eines Knotens es sich handelt, da alle Elemente immer nur ein übergeordnetes Element haben können. Die Funktion `domxml_parent()` ist das nicht-objektorientierte Gegenstück zu der Methode `parent()`. Sie hat dieselbe Wirkung:

```
$parent_node = domxml_parent($node[1]);
```

Das folgende Skript zeigt den Effekt dieser Methode/Funktion in einem Programm:

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root=domxml_root($doc);

$node = $root->children();
$parent_node = $node[1]->parent();
echo $parent_node->name;
?>
```

Listing 15.19: Die Methode parent()

Ausgabe:

```
datenbank
```

Das Skript bedarf eigentlich keiner großen Erklärung, da der Vorgang sehr simpel ist. Im ersten Schritt wird ein Objekt vom Root-Element erzeugt,

das wiederum ein Array mit allen Unterobjekten erschafft. Eines dieser Unterobjekte erzeugt ein weiteres Objekt, das wieder auf das Elternelement zeigt. Der Name dieses Elements ist natürlich »datenbank«, also wieder das Root-Element des Dokuments.

Eine weitere recht nützliche Methode aller Knotenobjekte ist `lastchild()`. Die Methode erlaubt es, direkt das letzte untergeordnete Objekt eines Knotens auszulesen und zurückzugeben. Die Methode ist vor allem dann nützlich, wenn eine XML-Datei regelmäßig aktualisiert wird und die neuesten Informationen immer am Ende der Datei zu finden sind.

```
$last_node =$root->lastchild();
```

Die Funktion `domxml_lastchild()` hat dieselbe Aufgabe und gibt ebenfalls das letzte untergeordnete Element eines Knotens zurück.

```
$last_node = domxml_lastchild($root);
```

Auf das Beispiel XML-Dokument bezogen würde dieser Aufruf das letzte Unterelement des Root-Elements auslesen.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);
$root=domxml_root($doc);

$node =$root->lastchild();
echo $node->content;
echo "<br>";
echo $node->type;
echo "<br>";
echo $node->getattr("num");
?>
```

Listing 15.20: Die Methode lastchild()

Ausgabe:

```
BillGatesEntwicklerBillGates@Microsoft.comhttp://www.micro-
soft.comhttp://www.billgates.de
1
5
```

Der Methodenaufruf bezieht sich logischerweise auf den Datensatz Nummer 5 im XML-Dokument. Dieser speichert die Daten von Bill Gates, die so einfach und bequem geparsed werden können.

15.6.4 Ein XML-Dokument mit DOM verändern

Der große Vorteil vom *Document Object Model* ist die Möglichkeit, neben dem Parsen von Dokumenten direkt in die Struktur des XML-Baums eingreifen zu können. DOM erlaubt es, vorhandene XML-Dokumente zu verändern oder sogar komplett neu zu erschaffen. Für den Zugriff auf die Daten und Struktur eines XML-Dokuments stehen eine Reihe von Methoden und Funktionen zur Verfügung, die mit dem bekannten XML-Objekt zusammenarbeiten.

Für einige der folgenden Möglichkeiten gibt es in der Regel auch eine nicht objektorientierte Funktion, die dieselbe Aufgabe erledigt.

Methode / Funktion	Beschreibung
`domxml_new_xmldoc()`	erzeugt ein neues XML-Objekt
`add_root()`	erzeugt ein neues Root-Element für ein Dokument. Vorhandene Daten werden überschrieben
`domxml_add_root()`	wie die Methode `add_root()`
`new_child()`	erzeugt einen neuen untergeordneten Knoten für das aktuelle Objekt
`domxml_new_child()`	wie die Methode `new_child()`
`setattr()`	setzt ein neues Attribut für einen Knoten
`domxml_setattr()`	wie die Methode `setattr()`
`set_content()`	legt den Inhalt eines Tags fest
`domxml_set_content()`	wie die Methode `set_content()`

Tabelle 15.15: Methode für den Schreibzugriff auf XML-Dokumente

Die einfachste Methode aus dieser Tabelle ist `add_root()`, die ein neues Root-Element für ein XML-Dokument definiert. Die Methode bekommt nur einen Parameter übergeben, der den Namen des neuen Elements definiert. Die Methode kann nur auf das Dokumentenobjekt ausgeführt werden.

```
$doc->add_root("home");
```

Der Effekt dieser Methode besteht darin, dass der komplette XML-Baum im Objekt aus dem Speicher gelöscht und ein neues Root-Element mit dem übergebenen Namen angelegt wird. Die Methode ist also nur für die

komplette Neudefinition eines XML-Dokuments sinnvoll, da andernfalls alle Daten verloren gehen würden.

```
<?
$dokument = implode("", file("data.xml"));
$doc = xmldoc($dokument);

$doc->add_root("home");
echo htmlentities($doc->dumpmem());
?>
```

Listing 15.21: Die Methode add_root()

Ausgabe:

```
<?xml version="1.0" encoding="ISO-8859-1"?> <home/>
```

Im Beispiel wird die bekannte XML-Datei data.xml geöffnet und ausgelesen. Die Daten stehen also im Speicher zur Verfügung und könnten genutzt werden. Im Skript wird aber die Methode `add_root()` aufgerufen, die ein neues Root-Element mit dem Namen »home« erzeugt. Die darauf folgende Ausgabe des Speichers zeigt, dass nun außer diesem Startelement keine weiteren Daten mehr vorhanden sind.

Das Dokument »data.xml« bleibt von diesem Vorgang unberührt! Bisher haben Sie nur die Daten im Speicher verändert. Wenn Sie die bearbeiteten Datensätze wieder speichern wollen, dann ist es notwendig, die Stringausgabe von `dumpmem()` in eine Datei zu schreiben. Diesen Vorgang werden wir am Ende des Kapitels besprechen.

Genau denselben Effekt wie bei der Methode `add_root()` haben Sie mit der Funktion `domxml_add_root()`.

```
domxml_add_root($doc, "home");
```

Neben dem Namen des neuen Elements muss hier noch das Zielobjekt als Parameter angegeben werden.

Ein leeres XML-Objekt erzeugen

Die Methode `xmldoc()` ist nicht in der Lage, ein XML-Objekt aus einem Leerstring zu erzeugen, ohne eine Fehlermeldung zu provozieren. Möchte man trotzdem ein neues XML-Dokument erschaffen, ohne ein anderes Dokument überschreiben zu müssen, braucht man die Funktion `domxml_new_xmldoc()`.

Genau wie `xmldoc()` gibt die Funktion ein Objekt zurück, auf das alle bekannten Methoden angewendet werden können. Als einzigen Parameter erwartet die Funktion einen String, der einem XML-Baum enthalten kann, aber nicht muss.

```
$doc = domxml_new_xmldoc("");
```

Ist der String leer, wird einfach ein leeres Dokument erstellt, ansonsten wird der XML-String in den Speicher eingelesen.

Neue Knoten erstellen

Die Methode `new_child()` arbeitet nach demselben Prinzip wie `add_root()`, mit dem Unterschied, dass `new_child()` es erlaubt, Elemente zu bestehenden Daten hinzuzufügen. Die Methode kann für alle Knotenobjekte angewendet werden und erzeugt einen untergeordneten Knoten, dem ein Name und ein bestimmter Inhalt zugewiesen wird.

```
$root -> new_child("element", "Ich bin Inhalt");
```

Der erste Parameter legt den Namen des neuen Elements fest. Der zweite Parameter übergibt den Inhalt, der diesem Tag zugewiesen wird. Um ein Containerelement zu erschaffen, muss anstelle des zweiten Parameters ein Leerstring übergeben werden. Die Kombination von Daten und weiteren Elementen ist natürlich erlaubt.

Die XML-Funktion `domxml_new_child()` ist das Gegenstück zur oben vorgestellten Methode. Sie erlaubt es ebenfalls, Subelemente zu einem bestehenden Knotenpunkt zu erstellen.

```
domxml_new_child($root,  "element", "Ich bin Inhalt");
```

Das Zielobjekt muss neben den benötigten Daten als zusätzlicher Parameter übergeben werden. Technisch besteht ansonsten kein Unterschied zur Methode `new_child()`.

```
<?
$doc = domxml_new_xmldoc("");
$doc->add_root("home");
$root=$doc->root();

domxml_new_child($root,  "element", "Ich bin Inhalt");

echo htmlentities($doc->dumpmem());
?>
```

Listing 15.22: Neue Knoten mit der Funktion domxml_new_child()

Ausgabe:

```
<?xml version=""?>
<home>
<element>Ich bin der Inhalt</element>
</home>
```

Im Beispiel wird mit der Funktion `domxml_new_xmldoc()` ein neues XML-Objekt ohne Inhalt erschaffen. Über die Methode `add_root()` wird dann die Grundlage für das Dokument erstellt, indem ein neues Root-Element geöffnet wird. Anhand dieses ersten Elements kann jetzt die komplette Struktur des XML-Baums aufgebaut werden.

Ein erster Schritt wird in Zeile sechs mit der Funktion `domxml_new_child()` getan. Es wird unter dem Element »home« ein neues Element mit dem Name »element« konstruiert, das den Inhalt »Ich bin der Inhalt« zugewiesen bekommt.

Attribute anlegen und verändern

Durch Attribute ist es möglich, bestehende Elementknoten so zu erweitern, dass sie zusätzliche Informationen speichern können, ohne den XML-Baum unnötig aufzublähen. Die Methode `setattr()` erlaubt es, über ein Knotenobjekt neue Attribute zu erstellen und einem Element zuzuordnen.

```
$object -> setattr($name, $value);
```

Die Methode erwartet zwei Parameter: den Namen des neuen Attributs sowie den Wert, der dem Attribut zugeordnet werden soll. Das Attribut wird automatisch dem Objekt zugeordnet, das diese Methode aufgerufen hat.

Das Gegenstück zu `setattr()` ist die Funktion `domxml_setattr()`, die ebenfalls Attribute für ein bestehendes Knotenobjekt erstellt.

```
domxml_setattr($object, $name, $value);
```

Wie bei allen XML-Funktionen muss hier zusätzlich das Zielobjekt als Parameter übergeben werden, um die Funktion erfolgreich ausführen zu können.

```
<?
$doc = domxml_new_xmldoc("");
$doc->add_root("home");
$root=$doc->root();
```

```
domxml_new_child($root,  "element", "Ich bin der Inhalt");

$node=$root->children();

$node[0]->setattr("wert", "unbegrenzt");
domxml_setattr($node[0], "num", "1");

echo htmlentities($doc->dumpmem());
?>
```

Listing 15.23: Neue Attribute für ein Element

Ausgabe:

```
<?xml version=""?>
<home>
<element wert="unbegrenzt" num="1">Ich bin der Inhalt</element>
</home>
```

Das Skript baut auf den Grundlagen der vorgehenden Beispiele auf und zeigt, wie dem neu erstellten Element »element« zwei Attribute gegeben werden. Für das Attribut »wert« wurde die objektorientierte Version von `setattr()` verwendet, während für das zweite Attribut »num« die DOMXML-Funktion `domxml_setattr()` eingesetzt wurde.

Den Inhalt von Elementen nachträglich verändern

Im Abschnitt über die Methode `new_child()` bzw. die Funktion `domxml_new_child()` haben Sie gesehen, wie der Content eines neuen Knotens im XML-Dokument gesetzt werden kann. Soll der Inhalt eines Elements allerdings im Laufe des Programms dynamisch verändert werden, dann ist diese Technik ungeeignet, da Sie den Knoten nicht jedes Mal neu erstellen wollen.

Wie gewohnt bietet das DOMXML-Modul hier wieder zwei Möglichkeiten an, dieses Problem zu lösen. Auf der einen Seite haben Sie die Methode `set_content()`, mit der der Inhalt eines Tags geschrieben bzw. überschrieben werden kann.

```
$object -> set_content($content);
```

Die Methode ersetzt einfach den Inhalt des Knotenelements durch den Stringparameter `$content`. Auf der anderen Seite können Sie die Funktion `domxml_set_content()` benutzen, die genauso wie die Methode `set_content` arbeitet.

```
domxml_set_content($object, $content);
```

Der Inhalt des Zielobjekts, das als Parameter übergeben werden muss, wird durch den String überschrieben. Die Funktion kann nur auf Elementobjekte abgewendet werden.

```
<?
$doc = domxml_new_xmldoc("");
$doc->add_root("home");
$root=$doc->root();

domxml_new_child($root,  "element", "Ich bin der Inhalt");
$node=$root->children();
$node[0]->setattr("wert", "unbegrenzt");
domxml_setattr($node[0], "num", "1");

domxml_set_content($node[0], "Ein besserer Inhalt");

echo htmlentities($doc->dumpmem());
?>
```

Listing 15.24: Die Funktion domxml_set_content()

Ausgabe:

```
<?xml version=""?>
<home>
<element wert="unbegrenzt" num="1">Ein besserer Inhalt</element>
</home>
```

Das Skript ist im Prinzip genauso aufgebaut wie das letzte Beispiel. Der Unterschied liegt in der zusätzlichen Zeile mit dem Aufruf der Funktion `domxml_set_content()`. Diese Zeile zeigt, wie der bereits bestehende String »Ich bin der Inhalt« durch den neuen String »Ein besserer Inhalt« ersetzt wird.

Das Dokument speichern

Wie ich bereits am Anfang des Kapitels festgestellt habe, hat die Veränderung des Dokuments im Speicher keinen Einfluss auf die ausgelesene Datei auf dem Server. Wenn Sie die erfolgreich durchgeführten Änderungen an Ihrem XML-Dokument auch in Zukunft nutzen möchten, ist es also nötig, die Daten wieder in eine Datei zu schreiben. Dasselbe gilt natürlich für komplett neu erstellte XML-Dokumente, die keine Grundlage in einer bestehenden Datei haben.

Das wichtigste Instrument für die Speicherung der Daten ist natürlich die Methode `dumpmem()`. Sie erlaubt es, den Speicher auszulesen, und gibt den aktuellen XML-Baum zurück.

```
$doc -> dumpmem();
```

Die Methode wird ohne Parameter über ein Dokumentenobjekt aufgerufen und gibt einen String mit dem Inhalt des Speichers wieder. In den vorhergehenden Beispielen wurde die Methode immer im Zusammenhang mit der Funktion `htmlentities()` verwendet, die uns erlaubt hat, die Tagdaten im Browser sichtbar zu machen. Für die folgenden Beispiele ist das nicht mehr nötig, da die Daten jetzt in eine Datei geschrieben werden sollen.

Das folgende Skript zeigt, wie ein neu erstelltes XML-Dokument in einer Datei abgespeichert wird. Die dabei verwendeten Funktionen sollten aus dem Kapitel über Dateifunktionen bekannt sein.

```
<?
$doc = domxml_new_xmldoc("");
$doc->add_root("home");
$root=$doc->root();

domxml_new_child($root,  "element", "Ich bin der Inhalt");
$node=$root->children();
$node[0]->setattr("wert", "unbegrenzt");
domxml_setattr($node[0], "num", "1");
domxml_set_content($node[0], "Ein besserer Inhalt");

$XML_str=$doc->dumpmem();

$fh=fopen("data.xml", "w");
$bool=fwrite($fh, $XML_str);
if($bool) {echo "XML-Dokument wurde erfolgreich gespeichert!";}
fclose($fh);
?>
```

Listing 15.25: XML-Daten mit der Methode dumpmem() speichern

Das Skript basiert zum größten Teil auf den Beispielen der letzten Abschnitte. Die Daten werden in diesem Skript aber nicht ausgegeben, sondern in einer Variable abgespeichert. Diese Daten werden dann über die Funktion `fwrite()` in die Datei »data.xml« geschrieben. Der Screenshot zeigt die Ausgabe des Skripts sowie die erstellte Datei.

Abbildung 15.10: Die XML-Daten wurden gespeichert

15.6.5 Übungen

Ü

1. Erklären Sie den Unterschied zwischen dem SAX-Parser und dem DOM-Parser.
2. Warum ist es unter Umständen besser, das SAX-Modell zu verwenden?
3. Was gibt die Dokumenteigenschaft »standalone« zurück?
4. Was unterscheidet die DOM-Methoden von den DOM-Funktionen?
5. Was ist ein Knoten?
6. Was gibt die Methode `children()` zurück?
7. Was gibt die Methode `dumpmem()` zurück?

16 Die Arbeitsumgebung

Im folgenden Kapitel werden die notwendigen Werkzeuge für die Entwicklung von PHP-Applikationen mit der MySQL-Datenbankunterstützung vorgestellt. Sie werden Schritt für Schritt lernen, wie Sie die richtige Software auch auf einem einzelnen Rechner oder einem lokalen Server für ein privates Intranet installieren und einrichten können.

Für die Veröffentlichung von dynamischen Seiten im Internet oder einem Intranet ist es erforderlich, dass PHP entweder als CGI-Programm oder (was noch besser ist) als Apache-Webserver-Modul unterstützt wird. Außerdem ist es notwendig, dass eine MySQL-Datenbank auf dem Server installiert ist, wenn Sie die Features einer Datenbank nutzen wollen. Darüber hinaus ist es sinnvoll, ein Administrationstool für den Zugriff auf die Datenbank einzurichten, um den einfachen Zugriff auf die Daten jederzeit gewährleisten zu können.

Sind diese Voraussetzungen erfüllt, steht der Erstellung von dynamischen Seiten nichts mehr im Wege. Sollten Sie Ihre Programme auf einem entfernten Server laufen haben, benötigen Sie neben einem ASCII-Editor, für die Erstellung der Programme, noch einen FTP-Client, um die Daten auf den Server zu kopieren.

Selbstverständlich ist es nicht zwingend notwendig, dass Sie den Apache Webserver als HTTP-Server verwenden, aber es gibt einige gute Gründe dafür. Vorweg sei aber gesagt, dass sowohl PHP als auch MySQL (oder eine andere Datenbank) genauso auf anderen Umgebungen laufen und gute Ergebnisse erzielen. Der Punkt, der für Apache als Webserver spricht, ist der Kostenfaktor, denn die Software ist ein OpenSource-Projekt und damit kostenlos.

Dasselbe gilt auch für PHP und MySQL auf UNIX- oder Linux-Systemen, die Software ist frei verfügbar und kann jederzeit aus dem Internet bezogen werden. Sowohl PHP als auch MySQL unterliegen der GNU General Public Lisence und sind deshalb kostenlos.

Wie Sie bereits in den vorhergehenden Kapiteln gelernt haben, handelt es sich bei PHP um eine serverseitige Skriptsprache. Dies bedeutet, dass der Code nicht vom Browser interpretiert wird, wie es beispielsweise bei JavaScript der Fall ist, sondern durch einen Interpreter auf dem Server.

In der Praxis bedeutet dies, dass Sie eventuelle Fehler im Skript erst bemerken, wenn Sie die erstellten Seiten auf den Server kopiert und ausgeführt haben. Es ist klar, dass sich so ein Vorgehen nicht gerade positiv auf die Entwicklungsarbeit auswirkt.

Zu Test- und Entwicklungszwecken bietet es sich daher an, dass Sie auf einem lokalen Rechner die nötige Software installieren, um PHP-Skripte ausführen zu können. Unter diesen Umständen haben Sie die Möglichkeit, alle Entwicklungsschritte in Ruhe und vor allem ohne lästige Uploadkosten nachzuvollziehen.

In diesem Kapitel werden Sie lernen, wie Sie den Apache Webserver, den PHP-Interpreter und eine MySQL-Datenbank mit Administratorsoftware auf einem Windowsrechner installieren können. Da die meisten Heimrechner auf einem Microsoft-Betriebssystem laufen, werde ich mit dieser Kombination wahrscheinlich die größte Zielgruppe erreichen.

16.1 Kapitelübersicht

- Der Apache Webserver
- PHP
- Die MySQL-Datenbank

16.2 Der Apache Webserver

Der Apache Webserver basiert auf dem NCSA Webserver von Rob McCool, der lange Zeit im Internet als der meistgenutzte Server galt. Inzwischen ist diese Position (seit 1996) vom Apacheserver übernommen worden. Der Name »Apache« entwickelte sich im Laufe der Zeit aus einem Wortspiel bezüglich des NCSA-Webservers. »Apache« entstand aus einer Reihe von Updates (Patches), mit denen der NCSA-Server erweitert und so »a patchy server« wurde. Durch die Namensähnlichkeit zu »Apache« war der Name geboren.

Inzwischen gehört der Apacheserver zur meistgenutzten Servertechnologie im Internet und hat im Laufe der Zeit weit über die Hälfte der Marktanteile erobert. Die Software steht für alle gängigen Plattformen zur Verfügung und unterstützt inzwischen auch die Microsoft Betriebssysteme. Der wichtigste Vorteil, der für Apache spricht, ist die Möglichkeit, das beste-

hende System durch weitere Module (z.B.: PHP4) zu erweitern und so immer neue Möglichkeiten zu eröffnen.

16.2.1 Den Apache Webserver installieren

Der erste Schritt zum eigenen Webserver auf dem heimischen PC ist der Download der aktuelle Apachedistribution. Die neueste Version für WIN32-Betriebssysteme hat die Nummer 1.3.22 und kann unter dem folgenden Link im Internet kostenlos heruntergeladen werden.

http://www.apache.org/dist/httpd/binaries/win32/

Die Installationsdatei ist ein selbstausführendes Setup-Archiv, das für die erfolgreiche Installation den Microsoft Installer in der Version 1.1 oder höher benötigt. Sollte Ihnen dieses Tool fehlen, dann können Sie es kostenlos auf der Apache-Webseite oder bei *http://www.microsoft.com/de* herunterladen.

Das Installationsprogramm selbst ist relativ einfach zu bedienen: Nach dem Start des Setup-Programms präsentiert sich ein Menü, durch das man sich mit den »back«- und »next«-Tasten navigieren kann. Im ersten Schritt müssen Sie die Nutzungsbedingungen anerkennen, um fortfahren zu können (siehe Abbildung 16.1).

Klicken Sie auf »I accept the terms in the lisence agreement« und fahren Sie mit der »next«-Taste fort. Der nächste Screen zeigt die Readme-Datei des Programms mit Hinweisen und Tipps für die Benutzung. Ich empfehle Ihnen die Datei einmal in einer ruhigen Minute zu überfliegen.

Auf der folgenden Seite müssen die ersten Angaben zur Netzwerkumgebung des Servers gemacht werden (siehe Abbildung 16.2).

Das Programm verlangt neben dem Domainnamen und der Serverkennung auch auch eine E-Mail-Adresse des Administrators und, je nach Betriebssystem, Angaben zur Nutzerverwaltung unter Windows.

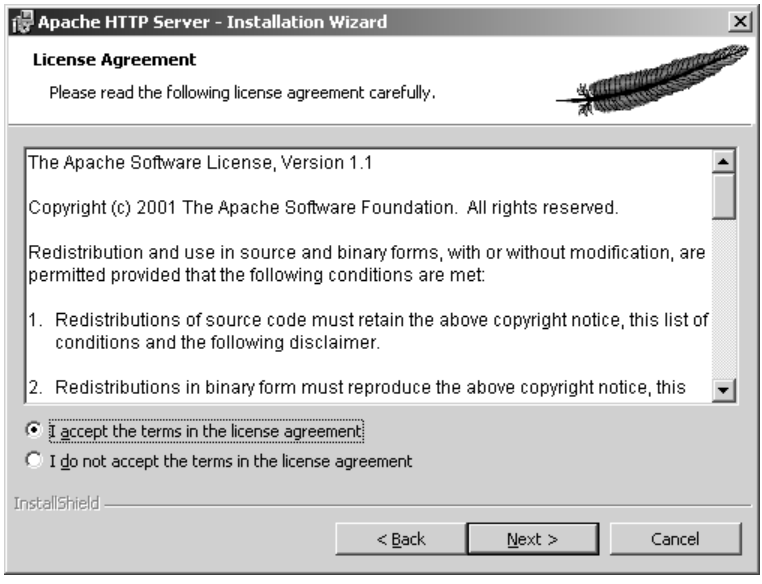


Abbildung 16.1: Nutzungsbedingungen des Apache

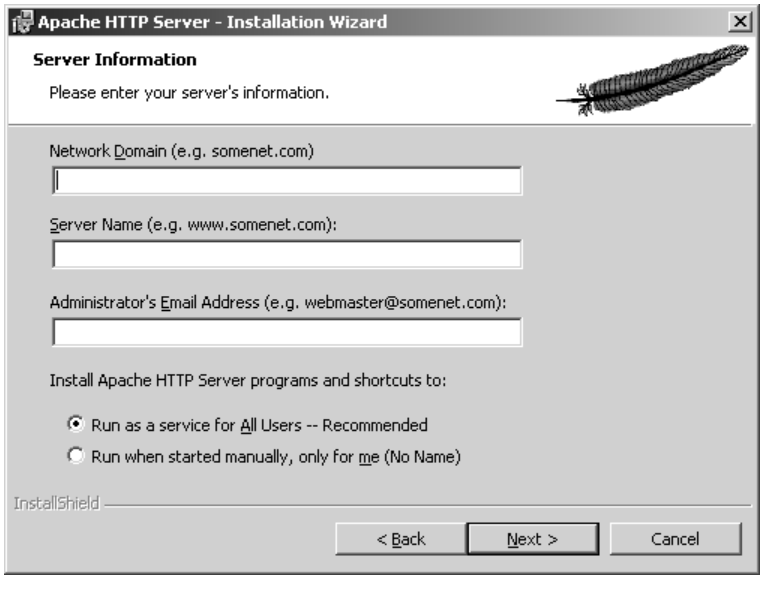


Abbildung 16.2: Angaben zum Server

Die nächsten Schritte legen die Art der Installation sowie das Zielverzeichnis auf dem Rechner fest, in das die Daten installiert werden sollen. Der Einfachheit halber sollten Sie die komplette Installation wählen und das Standardverzeichnis des Programms übernehmen.

```
C:\Programme\Apache Group\
```

So können Sie sicher sein, dass alle nötigen Pakete installiert werden und immer das richtige Verzeichnis benutzt wird.

Mit einem weiteren Klick auf »next« wird die Installation begonnen. Nach etwa einer Minute sollten Sie eine Erfolgsmeldung zu sehen bekommen, die Sie darüber aufklärt, dass die Installation erfolgreich verlaufen ist.

Ein Blick in den Explorer zeigt, dass ein neues Verzeichnis angelegt worden ist, in das alle Programmdateien des Apache Webservers installiert wurden. Unter `C:\Programme\Apache Group\Apache` finden Sie eine Reihe von weiteren Verzeichnissen, die für die verschiedenen Aufgaben des Servers bereitstehen. Die wichtigsten Verzeichnisse und deren Aufgabe sind hier kurz erklärt:

- `bin`: Das Verzeichnis bin enthält alle ausführbaren Binärdateien, die für den Betrieb des Webservers nötig sind. Die einzige Ausnahme bildet die Datei »Apache.exe«, über die der Server gesteuert wird. Wegen ihrer Wichtigkeit finden Sie die Datei im Wurzelverzeichnis der Applikation.
- `cgi-bin`: Der Ordner cgi-bin enthält alle Scriptfiles, die über das Common Gateway Interface genutzt werden können. Falls Sie PHP als CGI-Programm installieren, müssen Sie hier alle PHP-Skripte ablegen, damit sie ausgeführt werden können. Ein weiteres typisches Beispiel für die Verwendung des cgi-bin-Verzeichnisses ist der Einsatz von Perl-Skripten auf dem Server.
- `conf`: Das Verzeichnis `conf` enthält die Konfigurationsdateien des Apacheservers.
- `htdocs`: Der Name `htdocs` steht für Hypertext Documents. Der gleichnamige Ordner enthält also alle Dateien, die durch den Webserver im Internet veröffentlicht werden sollen. Dieses Verzeichnis ist das Wurzelverzeichnis des Webservers bei einem HTTP-Zugriff; hier werden alle HTML-Dokumente und natürlich PHP-Skripte abgelegt.

- `logs`: Wie der Name schon sagt, enthält dieses Verzeichnis die Logfiles des Servers. Das Verzeichnis enthält zwei Dateien, einmal die Zugriffsberichte in der Datei acess.log und auf der anderen Seite die Fehlermeldungen in der Datei error.log. Beide Dateien sind Textdateien, die mit einem einfachen Texteditor betrachtet werden können. Jeder Zugriff auf den Server wird in der Datei acess.log vermerkt, wobei jede Zeile einem Zugriff entspricht. Tritt ein Fehler auf, dann wird dieser in der Datei error.log vermerkt.

16.2.2 Den Apache Webserver konfigurieren

Nach der Installation des Apache Webservers ist das Programm im Prinzip lauffähig und kann ohne weiteres genutzt werden. Wollen Sie trotz allem ein paar Feineinstellungen an der Apachekonfiguration vornehmen, dann müssen Sie die Datei `httpd.conf` im Verzeichnis `C:\Programme\Apache Group\Apache\conf` editieren. Sie enthält alle Angaben, die für die Funktionen und den korrekten Ablauf des Programms nötig sind.

Bevor Sie Änderungen an der Datei vornehmen, empfehle ich Ihnen ein Backup der Daten zu erstellen, damit Sie immer auf eine lauffähige Version der Datei zurückgreifen können. Im Folgenden werde ich Ihnen einige wichtige Einstellungen der Datei vorstellen, die in ihrer Standardeinstellung korrekt funktionieren (zumindest für einen lokalen Entwicklungsserver). Im Laufe der weiteren Kapitel werden Sie allerdings einige Änderungen an der Konfiguration vornehmen müssen, darum kann es nicht schaden, schon jetzt einen Blick auf die Datei zu werfen.

Der größte Teil der Datei `httpd.conf` besteht aus Kommentaren, die mit dem #-Zeichen eingeleitet werden. Bevor Sie etwas ändern, lesen Sie bitte immer die Kommentare und vergewissern Sie sich, dass es sich um die richtige Zeile handelt.

- `ServerType`: Die Angabe hinter dem Schlüsselwort ServerType legt fest, um was für einen Server es sich handelt. Wenn der Server in keinem UNIX- Verbund arbeitet, ist hier nur der Wert »standalone« gültig.
- `ServerRoot`: Diese Angabe legt fest, in welchem Verzeichnis auf dem lokalen Rechner die Serversoftware installiert ist.
- `TimeOut`: Legt fest, nach wie viel Sekunden der Server eine Verbindung abbrechen soll.

- `KeepAlive`: Legt fest, ob persistente Verbindungen erlaubt sind oder nicht. Die folgenden Schlüsselwörter im Konfigurationsfile legen das Verhalten des Servers fest, wenn dauerhafte Verbindungen gestattet wurden.
- `Listen`: Erlaubt es, den Server an bestimmte Ports oder IP-Adressen zu binden. Sollte auskommentiert sein.
- `Port`: Legt fest, welchen Port der Server für den HTTP-Transfer nutzen soll. Der Standardport ist 80.
- `ServerAdmin`: Übergibt die E-Mail-Adresse des Webmasters. Hier sollte die E-Mail-Adresse stehen, die bei der Installation angegeben wurde.
- `ServerName`: Legt den Namen oder die IP-Adresse fest, unter welcher der Server im Netzwerk erreichbar ist. Wenn Sie den Server lokal nutzen, sollte hier »localhost« oder 127.0.0.1 stehen.
- `DocumentRoot`: Hier wird das Verzeichnis übergeben, in dem die Webdokumente abgelegt werden. Standardmäßig ist dies »C:/Programme/Apache Group/Apache/htdocs«. Sollten Sie das Verzeichnis ändern, dann müssen Sie diesen Eintrag aktualisieren.
- `AccessFileName`: Dieses Schlüsselwort legt den Namen der Konfigurationsdateien fest, über die einzelne Verzeichnisse des Servers verwaltet werden können.
- `DefaultType`: Hier wird Standard MIME-Type von Dokumenten festgelegt, die vom Server nicht identifiziert werden können. Ein guter Wert ist text/plain.
- `ErrorLog`: Legt den Namen und die Position der Logdatei für Fehler fest.
- `CustomLog`: Legt den Namen und die Position der Standard-Logdatei für Zugriffe fest.

16.2.3 Den Apache Webserver starten und stoppen

Nach der erfolgreichen Installation und Konfiguration des Servers wird es nun Zeit, das Programm zum ersten Mal zu starten. Da wir unter Windows arbeiten, ist der Vorgang relativ einfach, weil das Installationstool einen praktischen Shortlink in das Startmenü integriert hat.

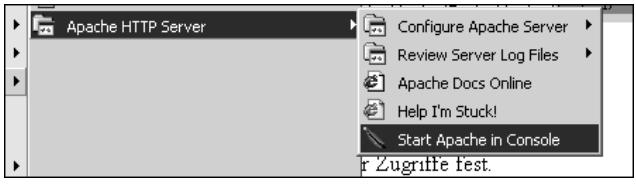


Abbildung 16.3: Verknüpfung im Startmenü

Der Link heißt »Start Apache in Consule« und ruft von der Eingabeoberfläche die Apachesoftware auf. Sobald die Meldung

```
Apache/1.3.22 (Win32) running...
```

erscheint, wurde der Server erfolgreich gestartet. Die Blackbox muss im Hintergrund weiterlaufen und darf erst wieder beim Stoppen des Servers geschlossen werden.

Abbildung 16.4: Der Apacheserver in der Konsole

Eine weitere Möglichkeit, den Apacheserver zu kontrollieren, ist direkt von der Eingabekonsole des Rechners. Öffnen Sie ein Kommandozeilenfenster und wechseln Sie in das Verzeichnis `C:\Programme\Apache Group\Apache`. Dort können Sie über das Programm `apache.exe` den Server Starten und stoppen.

- `apache start`: Startet den Server. Entspricht dem Befehl über das Startmenü von Windows.
- `apache stop`: Stoppt einen laufenden Server.
- `apache restart`: Startet einen laufen Server neu. Eventuelle Änderungen an der Konfiguration werden neu eingelesen.

Der einfachste und schnellste Weg, den Server stoppen, ist allerdings die Möglichkeit das Kommandozeilenfenster einfach zu schließen. Die Software wird sich möglicherweise beim nächsten Start beschweren, dass sie nicht sauber heruntergefahren wurde, aber das hat keine Auswirkungen auf das Programm.

16.2.4 Webseiten lokal anzeigen

Wenn der Apacheserver fehlerfrei gestartet wurde, sollten Sie ohne Probleme in der Lage sein, die Startseite des Apache aufzurufen. Starten Sie Ihren Browser und geben Sie folgenden Pfad in die Adresszeile ein:

```
http://localhost/
```

Sollten wider Erwarten Probleme auftreten, probieren Sie anstelle von »localhost« die IP-Adresse 127.0.0.1 bzw. den Namen des Servers, den Sie in der httpd.conf-Datei hinter dem Schlüsselwort »ServerName« angegeben haben. Falls Sie einen von der Standardeinstellung (Port 80) abweichenden Wert für »Port« angegeben haben, muss dies mit einem Doppelpunkt hinter dem Servernamen angegeben werden.

Abbildung 16.5: Es klappt! Der Apache läuft.

```
http://localhost:8080/
```

Wenn alles geklappt hat, sollte der Browser nun die Startseite des Apache Webservers anzeigen.

Von dieser Seite können Sie eine Reihe von Informationen über den Apache Webserver abrufen, wie zum Beispiel die Online-Dokumentation zur Software.

Wie Sie sich sicher schon denken, besteht diese Startseite aus nichts anderem als einer Reihe von HTML-Seiten, die mit der Installation in das htdocs-Verzeichnis des Servers kopiert wurden. Wenn Sie also eigene HTML-Seiten über den Webserver veröffentlichen wollen, brauchen Sie lediglich diese Dateien durch ihre eigenen zu ersetzen. Alternativ besteht natürlich die Möglichkeit ein Unterverzeichnis anzulegen.

16.3 PHP

Nachdem der Server erfolgreich installiert wurde, können Sie sich nun um die Einrichtung von PHP kümmern. Da PHP einen Webserver als Grundlage benötigt, muss die Installation in dieser Reihenfolge ausgeführt werden. Die aktuellsten Versionen von PHP3 und PHP4 können im Internet sowohl als Quellcode sowie als WIN32-Installationspaket kostenlos heruntergeladen werden. Da PHP der GNU General Public License unterliegt, darf die Software auch für kommerzielle Zwecke frei genutzt werden.

Die neueste Version von PHP finden Sie immer unter dieser Adresse im Internet:

http://www.zend.com/zend/download-php.php

Falls Sie den Sourcecode von PHP nichts selbst kompilieren wollen, empfehle ich Ihnen, das Win32-Installationspaket herunterzuladen.

16.3.1 PHP installieren

PHP kann in zwei verschiedenen Versionen auf einem Webserver verwendet werden: einmal als CGI-Programm (Common Gateway Interface) oder als Apache Modul. Als CGI-Programm arbeitet PHP mit jedem Webserver zusammen, die Modulversion ist nur auf dem Apache möglich.

CGI versus Apache Modul

Wird PHP als CGI-Programm installiert, dann liegen alle Programme im CGI-BIN Verzeichnis des Servers. Bei jeder HTTP-Anfrage, die der Webserver als PHP-Skript erkennt, muss der Rechner einen neuen Prozess für die Ausführung des Skriptes starten. PHP wird also mit jedem Start komplett neu geladen und nach der Ausführung wieder beendet.

Im Gegensatz dazu ist die modulare Variante beim Apacheserver ständig als Unterprogramm präsent und muss nicht mit jedem neuen Start initialisiert werden. Der Interpreter ist ständig bereit, ankommende Anfragen zu bearbeiten, ohne dabei den zeitaufwändigen Start und Stopp der Software ausführen zu müssen. Da auf diese Weise die Arbeitszeit sowohl für die Initialisierung als auch für das ordnungsgemäße Beenden von PHP eingespart wird, ist die modulbasierte Variante von PHP bei weitem die bessere Wahl, da eine vielfach höhere Arbeitsgeschwindigkeit erreicht wird.

PHP4 als Apachemodul installieren

Die Installation von PHP 4 unter Windows ist sehr einfach gehalten und bedarf keines großen Aufwands. Im ersten Schritt entpacken Sie das ZIP-File der aktuellen Binärdistribution in ein Verzeichnis Ihrer Wahl, beispielsweise

```
c:\php4\
```

Anschließend ändern Sie den Namen der Datei `php.ini-dist` in `php.ini`, und erstellen so die Initialisierungsdatei für den PHP-Interpreter. Ich empfehle Ihnen eine Sicherheitskopie der Datei anzulegen, damit Sie immer Zugriff auf die Standardeinstellungen der Installation haben, falls Probleme auftreten sollten.

Öffnen Sie die neue Datei `php.ini` mit einem beliebigen Texteditor und suchen Sie nach folgender Zeile:

```
extension_dir =
```

Hier muss der Pfad angegeben werden, in dem zukünfige PHP-Erweiterungen gespeichert werden sollen. Es empfiehlt sich, den Installationspfad des PHP-Interpreters anzugeben.

```
extension_dir = c:\php4
```

Speichern Sie das Dokument und suchen Sie anschließend nach der Zeile:

```
doc_root =
```

Dieses Schlüsselwort gibt an, in welchem Verzeichnis die PHP-Skripte auf dem Server abgelegt werden. In der Regel sollten Sie hier das htdocs-Verzeichnis des Webservers angeben, in dem auch die HTML-Seiten einer Webpage gespeichert werden.

```
doc_root = c:\Programme\Apache Group\Apache\htdocs
```

Alternativ ist es auch möglich ein eigenes Verzeichnis für PHP-Skripte auf dem Rechner anzugeben, was allerdings für einen Developmentserver sehr umständlich ist. Weitere Informationen zur php.ini finden Sie im Laufe des Kapitels.

Konfiguration vom Apache Webserver für PHP4

Um mit PHP unter dem Apacheserver arbeiten zu können sind einige Einstellungen in der Konfigurationsdatei vorzunehmen. Im ersten Schritt ist es nötig, den Server herunterzufahren, um die httpd.conf-Datei bearbeiten zu können. Sie finden die Datei im conf-Verzeichnis der Apachesoftware auf Ihrem Rechner. In der Standardinstallation lautet der Pfad wie folgt:

```
c:\apache group\apache\conf\httpd.conf
```

Öffnen Sie die Datei und suchen Sie nach den folgenden Zeilen im Text, die in aller Regel schon auskommentiert vorhanden sein sollten.

```
ScriptAlias /php/ "c:/php4/"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

Entfernen Sie die Kommentarzeichen und machen Sie den Eintrag für den nächsten Start des Servers ausführbar. Sollten die Zeilen in der Konfigurationsdatei fehlen, dann fügen Sie sie einfach an einer beliebigen Stelle ein.

Die drei Zeilen teilen dem Webserver mit, dass ab sofort alle Dateien mit der Endung `.php` vom PHP-Interpreter ausgeführt werden müssen, bevor sie an den Browser geschickt werden. Gleichzeitig wird über die Zeilen `ScriptAlias` und `Action` festgelegt, wo der Server den PHP-Interpreter finden kann.

Sollten Sie das Installationsverzeichnis also anders genannt haben, dann müssen Sie den Pfad in der Konfigurationsdatei anpassen. Ist das geschehen, muss der Server neu gestartet werden und PHP sollte einsatzfähig sein.

16.3.2 PHP-Programme lokal ausführen

Nach der Installation können PHP-Skripte genau wie normalen HTML-Seiten einfach in ein beliebiges Verzeichnis unterhalb des Veröffentlichungsverzeichnisses htdocs vom Apacheserver kopiert werden, um sie über den Webserver aufzurufen.

Ist PHP als Apachemodul installiert, werden alle Dateien mit der richtigen Dateiendung automatisch vom PHP-Interpreter ausgeführt, bevor sie an den anfragenden Clientrechner zurückgeschickt werden. Die Dateiendung, für die dieses Procedere gilt, wird in der Konfigurationsdatei httpd.conf festgelegt (`AddType`).

Starten Sie also den Webserver über das Startmenü und öffnen Sie ein Browserfenster, um das Skript aufrufen zu können. Dann legen Sie eine neue Textdatei an, die beispielweise den Namen test.php bekommt. Öffnen Sie die Datei und schreiben Sie dieses kurze Skript:

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

Listing 16.1: Beispielskript für den Test des PHP-Interpreters

Nach dem Abspeichern und Schließen kopieren Sie das Skript in das htdocs-Verzeichnis des Webservers. Jetzt ist das Skript veröffentlicht und kann über den Browser aufgerufen werden. Geben Sie folgende Adresse in die Adresszeile des Browser ein:

```
http://localhost/test.php
```

Bestätigen Sie mit Enter und warten Sie, bis die Seite geladen wurde. Da alle Informationen auf dem lokalen Rechner liegen, wird dieser Vorgang schnell passiert sein. Nun sollte sich Ihnen folgendes Bild zeigen (siehe Abbildung 16.6).

Ist das bei Ihnen der Fall? Herzlichen Glückwunsch, Sie haben PHP erfolgreich auf Ihrem System installiert. Wenn Sie eine Fehlermeldung bekommen, dann prüfen Sie, ob die Adresse stimmt oder ob Ihnen ein Tippfehler unterlaufen ist. Vielleicht haben Sie auch die Konfigurationsdatei fehlerhaft editiert. Überprüfen Sie auch, ob der Apacheserver nach der Installation von PHP richtig gestartet ist.

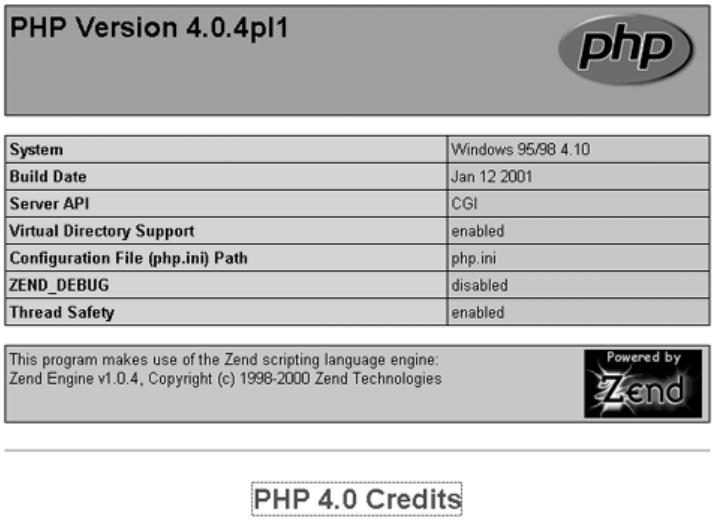


Abbildung 16.6: Die Funktion phpinfo()

Wenn alles geklappt hat, sollte Ihnen Ihr Browser eine umfangreiche Seite anzeigen, die Ihnen detaillierte Informationen über den Apache-Webserver sowie die PHP-Installation ausgibt.

16.3.3 Die Initialisierungsdatei php.ini

Nachdem Sie den PHP-Interpreter installiert haben, sind alle PHP-Skripte im Prinzip lauffähig. Sie können also alle Programme, die in diesem Buch besprochen wurden in das htdocs-Verzeichnis kopieren und über den Browser ausführen. Ausnahmen gelten für die Programme, die spezielle Zusatzmodule verwenden, die extra nachinstalliert werden müssen. Entsprechende Hinweise zur Installation finden Sie immer vor dem jeweiligen Kapitel.

Jedes Erweiterungsmodul muss in der Initialisierungsdatei php.ini bekannt gemacht werden, ehe Sie es nutzen können. Neben diesem Feature können Sie allerdings noch eine ganze Reihe weiterer Einstellungen vornehmen, die großen Einfluss auf das Verhalten des PHP-Interpreters haben.

Die Syntax

Die Syntax der php.ini ist recht einfach und ähnelt in groben Zügen dem Aufbau der Apache Konfigurationsdatei. Es gibt Kommentare, die durch ein Semikolon (;) am Anfang der Zeile markiert werden, und es gibt die bekannten Schlüssel-Wert-Zuweisungen, die die Optionen des Interpreters konfigurieren. Zusätzlich wird jeder Abschnitt der php.ini durch einen Titel markiert, der in eckigen Klammern angegeben wird.

Genau wie in der Sprachsyntax unterscheidet PHP zwischen Groß- und Kleinschreibung in der Initialisierungsdatei. Achten Sie also darauf, dass bei der Benennung der Optionen kein Fehler unterläuft. Häufig verwendete Werte sind On, Off, 1, 0, True, False, Yes oder No. Diese Werte unterliegen keiner bestimmten Schreibung und können beliebig angegeben werden.

Änderungen in der php.ini werden erst nach einem Neustart des Apache Webservers gültig. In den folgenden Zeilen werde ich die wichtigsten Aspekte und Konfigurationsmöglichkeiten der php.ini vorstellen.

Grundlegende Einstellungen

Der erste Abschnitt der php.ini betrifft grundlegende Einstellungen von PHP, die das Verhalten des Interpreters beeinflussen.

Schlüssel = Wert	Beschreibung
`engine = on`	Schaltet den Parser und Apache ein oder aus.
`short_open_tag = on`	Erlaubt den Kurz-Tag `<? ... ?>`. Ansonsten werden nur die Tags `<?php` und `<script>` erkannt.
`precision = 14`	Legt die Anzahl der Nachkommastellen fest, die berücksichtigt werden sollen.
`output_buffering = off`	Legt fest, ob die Ausgabe der Skripte in einen Buffer gespeichert werden soll, bevor sie ausgegeben wird. Das ermöglicht es, Header (inklusive Cookies) auch nach dem Inhalt zu senden. Die Engine wird allerdings verlangsamt.
`safe_mode = off`	Schaltet den sicheren Modus ein.
`disable_functions =`	Erlaubt es, Funktionen auszuschalten, die eventuelle Sicherheitslücken darstellen. Die Funktionsnamen werden durch Kommas getrennt angegeben.

Tabelle 16.1: Grundlegende Einstellungen

Schlüssel = Wert	Beschreibung

`highlight.string =`
`highlight.comment =`
`highlight.keyword =`
`highlight.bg =`
`highlight.default =`
`highlight.html =`
	Angaben zur Syntaxdarstellung von PHP.

`expose_php = on`	Legt fest, ob gezeigt werden soll, dass PHP auf dem Server installiert ist oder nicht.

Tabelle 16.1: Grundlegende Einstellungen (Forts.)

Angaben zur Leistung

Dieser Abschnitt übergibt Informationen zur Ausführungszeit und zur maximalen Größe eines Skriptfiles.

Schlüssel = Wert	Beschreibung
`max_execution_time =`	Maximale Ausführungszeit in Sekunden. Gilt nur unter UNIX.
`memory_limit =`	Maximale Skriptgröße

Tabelle 16.2: Angaben zur Leistung

Fehlerbehandlung

Dieser Abschnitt steuert das Fehlerverhalten und die Ausgabe des PHP-Interpreters.

Schlüssel = Wert	Beschreibung
`error_reporting =`	Legt fest, welche Fehler vom Parser ausgegeben werden sollen. Der Übergabewert wird aus folgenden Bit-Werten ermittelt: 1 = Normale Fehler 2 = Normale Warnungen 4 = Parser-Fehler 8 = Hinweise

Tabelle 16.3: Fehlerbehandlung

Schlüssel = Wert	Beschreibung
	Die Summe aller gewünschten Fehler wird als Wert dem Schlüssel übergeben. Alternativ können auch Konstanten übergeben werden, die bestimmte Fehlerkombinationen symbolisieren. Mehr Angaben dazu finden Sie in der php.ini.
`display_errors = on`	Legt fest, ob die Fehler im Browser erscheinen sollen oder nicht. Auf Produktionsservern sollte diese Feature ausgeschaltet werden.
`log_errors = off`	Falls aktiviert, werden die Fehler in der Error-Log-Datei des Webservers ausgegeben.
`error_prepend_string =`	Legt den HTML-Code fest, der vor einer Fehlermeldung erscheinen soll.
`error_append_string =`	Legt den HTML-Code fest, der nach einer Fehlermeldung erscheinen soll.
`error_log = filename`	Legt einen bestimmten Dateinamen fest, in dem die Fehler ausgegeben werden sollen.

Tabelle 16.3: Fehlerbehandlung (Forts.)

Einstellungen zur Datenbehandlung

Der folgende Abschnitt behandelt die Optionen, die den Umgang mit GET/POST – und Cookie-Variablen regeln. Außerdem besteht die Möglichkeit die Verarbeitung von Sonderzeichen zu automatisieren.

Schlüssel = Wert	Beschreibung
`variables_oder =`	Legt die Reihenfolge der Datenbehandlung fest.
`register_globals = on`	Legt fest, ob Übergabevariablen als globale Variablen automatisch im Skript zur Verfügung stehen. Falls ausgeschaltet, müssen alle Werte aus den Umgebungsvariablen `$HTTP_GET_VARS[]`, `$HTTP_POST_VARS[]` und `$HTTP_COOKIE_VARS[]` ausgelesen werden.
`post_max_size = 8M`	Legt die maximale Größe der Übergabedaten fest.
`magic_quotes_gpc =`	Falls eingeschaltet, werden alle Sonderzeichen in Übergabestrings mit Slashes versehen.

Tabelle 16.4: Datenbehandlung

Schlüssel = Wert	Beschreibung
magic_quotes_runtime=	Wie oben, nur für dynamisch erzeugte Daten.
default_mimetype =	Legt den Standard MIME-Type fest, der für PHP-Files gesendet werden soll.

Tabelle 16.4: Datenbehandlung (Forts.)

Pfade und Verzeichnisse

In diesem Abschnitt können Sie Pfade und Verzeichnisse für den PHP-Interpreter festlegen.

Schlüssel = Wert	Beschreibung
include_path =	Pfad für Include-Dateien von PHP.
doc_root =	Root-Verzeichnis der PHP-Skripte.
extension_dir =	Pfad für alle Erweiterungsmodule von PHP.
file_uploads = on	Schaltet die Möglichkeit des HTTP-Uploads ein oder aus.
upload_tmp_dir =	Legt das Verzeichnis für temporäre Upload-Dateien fest.
upload_max_filesize =	Legt die maximale Größe von Upload-Dateien fest.
allow_url_fopen = on	Erlaubt es, URLS wie Dateien zu öffnen.

Tabelle 16.5: Pfade und Verzeichnisse

Einbindung von Windowsmodulen

Einige Funktionen von PHP erwarten bestimmte Module, um korrekt funktionieren zu können. Jede Erweiterung muss unter Windows durch eine externe Bibliotheksdatei eingebunden werden. Informationen zu den einzelnen Modulen und deren Installation erhalten Sie in dem jeweiligen Kapitel.

Schlüssel = Wert	Beschreibung
extension=php_bz2.dll	bzip2-Kompression
extension=php_ctype.dll	ctype-Funktionen

Tabelle 16.6: Windowsmodule

Schlüssel = Wert	Beschreibung
extension=php_cpdf.dll	cpdf-Funktionen
extension=php_curl.dll	URL-Grabbing-Funktionen
extension=php_cybercash.dll	Cybercash-Zahlungsfunktionen
extension=php_db.dll	Datenbankfunktionen
extension=php_dba.dll	dba-Datenbank-Funktionen
extension=php_dbase.dll	dbase-Funktionen
extension=php_dbx.dll	database abstraktion extension
extension=php_domxml.dll	DOMXML-Funktionen
extension=php_dotnet.dll	.NET-Funktionen
extension=php_exif.dll	exif-Funktionen
extension=php_fdf.dll	PDF-Form-Funktionen
extension=php_filepro.dll	FilePro-Funktionen
extension=php_gd.dll	GD-Grafik-Funktionen
extension=php_gettext.dll	Textfunktionen
extension=php_hyperwave.dll	Hyperwave-Funktionen
extension=php_iconv.dll	iconv-Funktionen
extension=php_ifx.dll	Informix-Datenbankfunktionen
extension=php_iisfunc.dll	IIS-Funktionen
extension=php_imap.dll	IMAP-Mailfunktionen
extension=php_ingres.dll	INGRES2-Datenbankfunktionen
extension=php_interbase.dll	Interbase Datenbankfunktionen
extension=php_java.dll	Javaunterstützung
extension=php_ldap.dll	LDAP-Funktionen
extension=php_mcrypt.dll	mcrypt-Funktionen
extension=php_mhash.dll	mhash-Funktionen
extension=php_ming.dll	Ming-SWF-Funktionen
extension=php_mssql.dll	MSSQL-Datenbank-Funktionen
extension=php_oci8.dll	Oracle oci8-Funktionen

Tabelle 16.6: Windowsmodule (Forts.)

Schlüssel = Wert	Beschreibung
extension=php_openssl.dll	SSL-Funktionen
extension=php_oracle.dll	Oracle-Datenbank-Funktionen
extension=php_pdf.dll	PDF-Funktionen
extension=php_pgsql.dll	Postgre-Datenbankfunktionen
extension=php_printer.dll	Win32-Druckfunktionen
extension=php_snmp.dll	SNMP-Funktionen
extension=php_sybase_ct.dll	Sybase-Datenbankfunktionen
extension=php_yaz.dll	yaz-Funktionen
extension=php_zlib.dll	zlib-Funktionen

Tabelle 16.6: Windowsmodule (Forts.)

Weitere Einstellungen

In diesem Abschnitt werden verschiedene Einstellungen zu den einzelnen Modulen von PHP übergeben.

Schlüssel = Wert	Beschreibung
SMTP = localhost	Legt den SMTP-Server fest.
sendmail_from =	Standardabsender von E-Mails
sql.safe_mode = off	SQL wird im sicheren Modus betrieben.
mysql.allow_persistent = on	Persistente Verbindungen werden erlaubt.
mysql.max_persistent = -1	Maximale Anzahl der persistenten Verbindungen. –1 bedeutet »keine Einschränkungen«
mysql.max_links = -1	Maximale Anzahl aller MySQL-Verbindungen.
mysql.default_port =	Standardport für MySQL-Verbindungen.
mysql.default_socket =	Standardsocket für MySQL-Verbindungen.
mysql.default_host =	Standardserver für MySQL-Verbindungen.
mysql.default_user =	Standarduser für MySQL-Verbindungen.
mysql.default_password =	Standardpasswort für MySQL-Verbindungen.

Tabelle 16.7: Weitere Einstellungen

Session-Verwaltung

Diese Angaben dienen zur Steuerung von Sessions unter PHP4.

Schlüssel = Wert	Beschreibung
`session.save_handler = files`	Angabe zur Datei, zum Speichern der Sessiondaten.
`session.save_path = /tmp`	Pfad zu den Sessiondaten.
`session.use_cookies = 1`	Aktiviert Cookies für die Sessionverwaltung.
`session.name = PHPSESSID`	Standardname für die Sessionverwaltung.
`session.auto_start = 0`	Legt fest, ob Sessions automatisch gestartet werden sollen.
`session.cookie_lifetime = 0`	Legt die Lebensdauer von Sessioncookies in Sekunden fest.
`session.cookie_path = /`	Legt den Pfad für die Gültigkeit des Cookies fest.
`session.cookie_domain =`	Legt die Domain für die Gültigkeit des Cookies fest.
`session.gc_maxlifetime =`	Maximale Lebensdauer von Sessiondaten in Sekunden.

Tabelle 16.8: Sessions

16.4 Datenbankserver MySQL

Für den Aufbau von dynamischen Webseiten mit aktuellen Daten und Angeboten ist der Einsatz einer Datenbanksoftware unumgänglich. Im Kapitel über PHP und Datenbanken wurde bereits ausführlich über die Vorteile und Einsatzmöglichkeiten einer Datenbank diskutiert. In diesem Kapitel werden Sie nun erfahren, wie Sie ganz einfach die WIN32-Version der MySQL-Datenbank auf Ihrem Rechner installieren und nutzen können.

MySQL liegt zur Zeit (Dezember 2001) in der aktuellen stabilen Version 3.23 vor und kann kostenlos unter der GNU GPL im Internet bezogen werden. Die Version 4.0 kann bereits als Alpha-Version getestet werden, ist aber für den professionellen Einsatz im Netz nicht empfehlenswert. Die stabile Version von MySQL finden Sie unter:

http://www.mysql.com/downloads/mysql-3.23.html

Wenn Sie keine DSL-Leitung oder einen ähnlichen Anschluss ans Internet haben, sollten Sie etwas Zeit mitbringen, denn die Installationsdatei ist 12 Megabyte groß. Wenn die Datei aber erst einmal heruntergeladen wurde, steht der Installation nichts mehr im Wege.

16.4.1 MySQL installieren

Die MySQL-Version 3.23 für Win32-Systeme wird als Zip-Datei im Internet zum Download bereitgestellt. Laden Sie die Datei auf Ihren Rechner und entpacken Sie das Installationsarchiv in ein temporäres Verzeichnis mit dem Namen

```
c:\mysql_inst\
```

Öffnen Sie das Verzeichnis und suchen Sie die Datei »setup.exe«. Es handelt sich dabei um eine typische Windows Installationsdatei, die mit einem entsprechenden Symbol gekennzeichnet ist. Starten Sie die Datei mit einem Doppelklick. Es öffnet sich ein Installationsmenü, in dem Sie den Pfad und die Art der Installation wählen können.

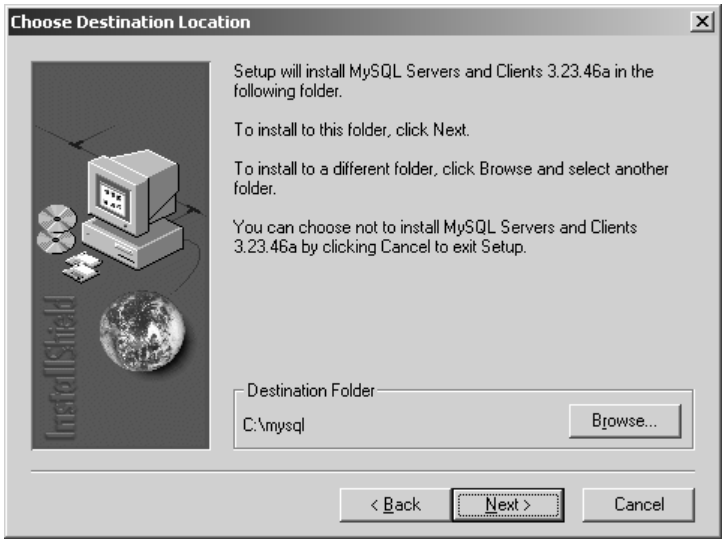


Abbildung 16.7: MySQL-Installation

Ich empfehle Ihnen den Standardpfad c:\mysql\ sowie die Installationsart »typical« zu übernehmen. Nach etwa einer Minute sollte die Installation abgeschlossen sein und der Datenbankserver zur Verfügung stehen.

Ist die Installation erfolgreich verlaufen, können Sie das temporäre Installationsverzeichnis von Ihrer Festplatte löschen. Es wird nicht mehr benötigt.

16.4.2 Verzeichnisse und Dateien

Ein Blick in den Explorer zeigt, dass ein neues Verzeichnis mit dem Namen »mysql« angelegt worden ist. Unter anderem sollten Sie folgende wichtigen Unterverzeichnisse in diesem Directory vorfinden:

Verzeichnisname	Beschreibung
`/data`	Datenverzeichnis für alle Daten, die in der Datenbank gespeichert werden.
`/bin`	Binärdateien für die Steuerung von MySQL.
`/docs`	Dokumentationen zu MySQL
`/bench`	SQL Benchmarks
`/examples`	Beispiele zur Arbeit mit MySQL
`/include`	Include-Dateien
`/lib`	Programmbibliotheken

Tabelle 16.9: Verzeichnisse im MySQL-Ordner

Im Unterverzeichnis `/bin` sind alle Programme und Tools abgelegt, die Sie für die Arbeit mit MySQL benötigen. Die nachfolgende Liste gibt Ihnen eine kurze Übersicht über die wichtigsten Programme und deren Aufgabe:

Programm	Beschreibung
`mysql.exe`	Clientprogramm für den SQL-Zugriff auf die Datenbanksoftware.
`mysqladmin.exe`	Programm zum Aufruf von Batchdateien mit SQL-Befehlen.

Tabelle 16.10: Wichtige Programme

Programm	Beschreibung
mysqldump.exe	Tool um Daten aus der Datenbank auf der Festplatte zu sichern (Dumpfile).
mysqlimport.exe	Programm um ASCII-Dateien in eine Tabelle einzulesen.
mysqlshow.exe	Zeigt alle installierten Datenbanken an.
mysqlcheck.exe	Überprüft die gespeicherten Daten.
winmysqladmin.exe	Startet den MySQL-Server unter Windows.
WINMYSQLADMIN.HLP	Hilfedatei zur MySQL-Software für Windows.
MySqlManager.exe	Windowsprogramm für die Arbeit mit der MySQL-Datenbank.

Tabelle 16.10: Wichtige Programme

Nach der Installation sind bereits zwei Datenbanken installiert, wie der Aufruf des Programms mysqlshow.exe beweist.

```
C:\mysql\bin>mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+
```

Die Datenbank »test« ist leer und wird von MySQL nur zu Testzwecken angelegt. Sie hat keine weitere Bedeutung für die Arbeit mit dieser Software. Im Gegensatz dazu ist die Datenbank »Mysql« sehr wichtig, denn sie speichert die Zugriffsrechte aller User, die auf diese Datenbank zugreifen. Auch wenn diese Informationen auf einen privaten Entwicklungsserver wahrscheinlich eher zweitrangig sind, werde ich sie in einem eigenen Kapitel kurz beleuchten.

16.4.3 MySQL starten und beenden

Der MySQL-Datenbankserver ist nach der Installation sofort einsatzbereit und kann ohne weiteren Konfigurationsaufwand genutzt werden. Die Windowsoberfläche macht den Umgang mit der Software sehr komfortabel, da über das Programm »winmysqladmin.exe« der einfache Administrationsaufwand erledigt werden kann.

Wenn Sie das Programm zum ersten Mal starten, wird MySQL versuchen die Konfigurationsdatei »my.ini« zu finden. Ist diese Datei nicht vorhanden, wird sich folgendes Konfigurationsfenster öffnen.

Abbildung 16.8: Ein User wird angelegt

Tragen Sie einen Benutzernamen und ein Passwort Ihrer Wahl in die Textfelder ein und klicken Sie auf OK. Die »my.ini«-Datei wird nun automatisch erstellt und ein neuer User wird in die Benutzertabelle von MySQL eingetragen. Die Konfigurationsdatei von MySQL wird automatisch in das Systemverzeichnis von Windows installiert.

Wenn alles geklappt hat, schließt sich das Konfigurationsfenster und ein MySQL-Icon wird sich in Form einer Ampel in der unteren rechten Symbolleiste von Windows öffnen. Steht die Ampel auf Rot, dann ist der MySQL-Server inaktiv und muss gestartet werden. Ein Rechtsklick auf die Ampel öffnet ein kleines Menü, über das Sie den Server starten und stoppen können.

Die Ampel sollte jetzt von Rot auf Gelb und dann auf Grün springen und signalisiert so, dass der Server aktiv ist. Jetzt können Sie mit allen MySQL-Tools auf die Datenbank zugreifen. Auch PHP-Verbindungen sind jetzt möglich. In Zukunft wird der MySQL-Server automatisch bei jedem Start von Windows aktiviert werden.

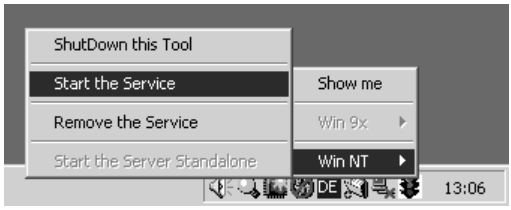


Abbildung 16.9: MySQL wird gestartet

Über die Schaltfläche »Show me« gelangen Sie zum Hauptfenster von »winmysqladmin.exe«. Hier haben Sie die komplette Übersicht über alle Daten und Vorgänge, welche die MySQL-Datenbank betreffen.

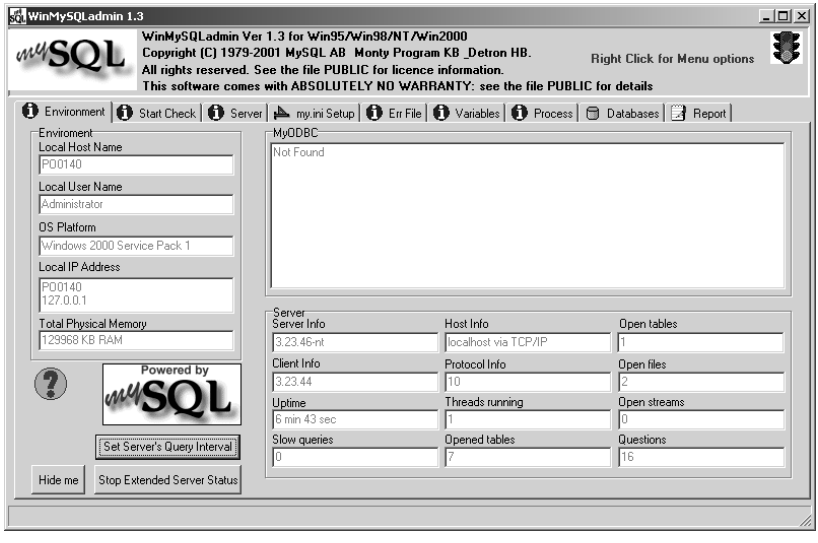


Abbildung 16.10: WinMySQLadmin

Sie haben Zugriff auf die Konfigurationsdatei »my.ini« sowie den vollständigen Überblick über alle Prozesse und bestehenden Datenbanken. Außerdem können Sie über den Menüpunkt »Databases« einen Blick in die Datenstrukturen aller Tabellen werfen. Ein Rechtsklick auf die große Ampel in der oberen rechten Ecke öffnet ein Menü, das es erlaubt, dieses Fenster wieder zu schließen.

16.4.4 Ein erster Test

Nachdem die Installation abgeschlossen ist, sollten Sie in einem ersten Test überprüfen, ob der Server ordnungsgemäß läuft. Achten Sie darauf, dass die Ampel auf Grün steht, bevor Sie den Test beginnen, da ansonsten der Server nicht aktiviert ist.

Rufen Sie das Programm »mysqladmin.exe« aus dem bin-Verzeichnis des Servers auf und übergeben Sie die Parameter »version«:

```
c:\mysql\bin\mysqladmin version
```

Läuft der Server ordnungsgemäß, dann sollten Sie beispielweise folgende Ausgabe erhalten:

```
C:\mysql\bin>mysqladmin version
mysqladmin  Ver 8.23 Distrib 3.23.46, for Win95/Win98 on i32
Copyright (C) 2000 mySQL AB & mySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free soft-
ware,
and you are welcome to modify and redistribute it under the GPL
license

Server version          3.23.46-nt
Protocol version        10
Connection              . via named pipe
UNIX socket             mySQL
Uptime:                 14 min 24 sec

Threads: 2  Questions: 123  Slow queries: 0  Opens: 8  Flush tables:
1  Open tables: 2 Queries per second avg: 0.142
```

Je nach Betriebssystem und Version kann sich die Ausgabe leicht verändern, aber im Aufbau sollte die Aussage identisch sein. Ist das bei Ihnen der Fall, dann lassen Sie sich gratulieren: Der MySQL-Server wurde erfolgreich installiert und ist einsatzfähig!

Zu guter Letzt möchte ich Ihnen noch ein kleines PHP-Skript vorstellen, das überprüft, ob die MySQL-Datenbank auch mit PHP erreicht werden kann. Kopieren Sie das Programm einfach in das htdocs-Verzeichnis des Webservers und rufen Sie es über den Browser auf.

```
<?
$server = "localhost";
$user = "root";
$pass = "geheim";
```

```
$dbh = mysql_connect($server, $user, $pass);
if($dbh) {echo "mySQL-Verbindung erfolgreich!<br>";}
$bool = mysql_select_db("mysql");
if($bool) {echo "Datenbank erfolgreich gewählt!<br>";}
$query = "select user from user";
$erg = mysql_query($query, $dbh);
echo "<br>User:<br>";
while ($data = mysql_fetch_row($erg))
{
   echo $data[0] . "<br>";
}
?>
```

Listing 16.2: Testprogramm für die MySQL-Datenbankverbindung

Ausgabe:

```
mySQL-Verbindung erfolgreich!
Datenbank erfolgreich gewählt!

User:
root
test
master
dirk
```

Das Programm gibt alle Usernamen aus, die Zugriff auf die MySQL-Datenbank auf Ihrem System haben. Wenn PHP erfolgreich die Daten auslesen kann, ist Ihr Entwicklungsserver korrekt eingerichtet und kann genutzt werden.

16.4.5 User verwalten

Die MySQL-Userverwaltung basiert auf einem relativ komplexen System von Berechtigungen, die Benutzern eines bestimmten Hosts die Rechte für verschiedene Befehle erteilen oder verweigern. Es wird also nicht zwischen einem generellen Lese- und Schreibrecht unterschieden, sondern zwischen den Möglichkeiten SQL-Befehle wie `INSERT`, `DELETE`, `CREATE`, `SELECT` oder `UPDATE` auszuführen.

Im Vergleich zu anderen passwortgeschützten Systemen bestehen die MySQL-Zugangsdaten nicht nur aus einem Usernamen und einem Passwort, sondern es wird auch differenziert zwischen den Zugriffen von unterschiedlichen Servern. Das bedeutet in der Praxis, dass ein User, der von System A auf die Datenbank zugreift, komplett anders behandelt werden kann als genau derselbe User, der mit denselben Zugangsdaten von System B zugreift.

Die Einstellungen der User- und Sicherheitspolitik von MySQL werden in der Datenbank »mysql« gespeichert.

Die Tabelle user

Die Tabelle »user« enthält User/Passwort/Host-Kombinationen, die Zugriff auf den MySQL-Server haben. In dieser Tabelle werden alle Berechtigungen für die verschiedenen SQL-Befehle gespeichert, die ein einzelner User von einem bestimmten Host besitzen kann. Die hier festgelegten Rechte gelten für alle Datenbanken auf dem MySQL-Server. Spezielle Rechte können über die Tabelle »db« festgelegt werden.

Die Tabelle »user« hat folgenden Aufbau:

Field	Type	Null	Key	Default	Extra
Host	char(60)		PRI		
User	char(16)		PRI		
Password	char(16)				
Select_priv	enum(»N«, »Y«)			N	
Insert_priv	enum(»N«, »Y«)			N	
Update_priv	enum(»N«, »Y«)			N	
Delete_priv	enum(»N«, »Y«)			N	
Create_priv	enum(»N«, »Y«)			N	
Drop_priv	enum(»N«, »Y«)			N	
Reload_priv	enum(»N«, »Y«)			N	
Shutdown_priv	enum(»N«, »Y«)			N	
Process_priv	enum(»N«, »Y«)			N	
File_priv	enum(»N«, »Y«)			N	
Grant_priv	enum(»N«, »Y«)			N	
References_priv	enum(»N«, »Y«)			N	
Index_priv	enum(»N«, »Y«)			N	
Alter_priv	enum(»N«, »Y«)			N	

Tabelle 16.11: Die Tabelle user

Jeder Datensatz steht für eine mögliche User/Host-Kombination, die Zugriff auf die MySQL-Datenbank hat. Das Passwort wird aus Sicherheitsgründen codiert in der Spalte »Password« gespeichert. Die folgenden Spalten enthalten die Zugriffsrechte für bestimmte SQL-Befehle auf dem Datenbankserver. Die Felder können entweder den Wert »N« für »Keine Berechtigung« oder den Wert »Y« für »Zugriff erlaubt« übergeben bekommen.

Die Tabelle db

In der Tabelle »db« wird im Detail festgelegt, welcher User mit einer bestimmten User/Host-Kombination in welcher Datenbank über welche Berechtigungen verfügt. Die Angaben gehen über die Einstellungen der Tabelle »user« hinaus und dienen dazu, Zugriffsrechte für bestimmte Tabellen festzulegen.

Die Tabelle »db« hat folgenden Aufbau:

Field	Type	Null	Key	Default	Extra
Host	char(60)		PRI		
Db	char(64)		PRI		
User	char(16)		PRI		
Select_priv	enum(»N«, »Y«)			N	
Insert_priv	enum(»N«, »Y«)			N	
Update_priv	enum(»N«, »Y«)			N	
Delete_priv	enum(»N«, »Y«)			N	
Create_priv	enum(»N«, »Y«)			N	
Drop_priv	enum(»N«, »Y«)			N	
Grant_priv	enum(»N«, »Y«)			N	
References_priv	enum(»N«, »Y«)			N	
Index_priv	enum(»N«, »Y«)			N	
Alter_priv	enum(»N«, »Y«)			N	

Tabelle 16.12: Die Tabelle db

In der Tabelle »db« können für jede User/Host-Kombination die Rechte bezüglich einer bestimmten Datenbank auf dem MySQL-Server festgelegt werden. Jeder Datensatz steht für eine bestimmte Datenbank, die im Feld

»Db« abgegeben wird. In den nachfolgenden Feldern können dann spezielle Rechte vergeben werden.

Soll der Zugriff unabhängig von einem bestimmten Host sein, dann ist es möglich, über den Platzhalter % generelle Rechte für einen User zu vergeben.

Ein Beispiel

Das folgende Beispiel zeigt, wie mit einfachen SQL-Befehlen ein neuer User für dem MySQL-Server angelegt wird. Der neue Benutzer bekommt den Usernamen »tester« und das Passwort »test« zugeteilt. Er soll die Möglichkeit haben vom Host »localhost« auf die Datenbanken »mysql« und »test« zugreifen zu können. Von allen anderen Rechnern soll er lediglich Leserechte (SELECT) für die Datenbank »test« bekommen.

Als Erstes wird der User in die Tabelle »user« geschrieben. Dabei wird zwischen dem Host »localhost« und allen anderen Zugriffsmöglichkeiten unterschieden.

```
INSERT INTO user VALUES ("localhost", "tester", password("test"),
"Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y");
INSERT INTO user (Host, User, Password, Select_priv) VALUES
("%","tester", password("test"), "Y");
```

Das Passwort wird über die Stringfunktion `passwort()` codiert. Im zweiten Schritt wird die Tabelle »db« angepasst, um die Zugriffsrechte des neuen Users von außerhalb dieses Systems weiter einzuschränken.

```
INSERT INTO db (Host, Db, User, Select_priv) VALUES ("%", "test",
"tester", "Y");
```

Der letzte Punkt ist die Anpassung der Rechte vom User »tester« auf dem Host »localhost«. Er soll nur auf die Datenbanken »mysql« und »test« zugreifen können.

```
INSERT INTO db VALUES ("localhost", "test", "tester", "Y",
"Y","Y","Y","Y","Y","Y","Y","Y","Y","Y");
INSERT INTO db VALUES ("localhost", "mysql", "tester", "Y",
"Y","Y","Y","Y","Y","Y","Y","Y","Y","Y");
```

Fertig. Jetzt ist das Zugriffsprotokoll für einen neuen User komplett. Natürlich ist die Vergabe von Rechten sehr blauäugig, da »tester« mit vollen Zugriffsrechten auf die Datenbank »mysql« ausgestattet ist. Er könnte also ohne Probleme seine Rechte großzügig erweitern. Achten Sie also darauf, dass niemand außer dem Administrator Zugriff auf die Datenbank »mysql« hat.

Teil III – Reference

Referenz

Diese Referenz enthält alle Befehle, die in dem Buch vorgestellt wurden. Sie soll Ihnen helfen, Informationen zu den Anweisungen ohne Probleme nachzuschlagen, ohne dabei das ganze Buch durchblättern zu müssen. Aus diesem Grund ist die Reihenfolge der Befehle alphabetisch geordnet, so dass das Suchen und Finden von Schlagwörtern so einfach wie möglich ist.

abs()

Die Funktion abs() berechnet den absoluten Betrag eines Wertes.

```
mixed abs(mixed zahl)
```

Als Parameter wird der Ausgangswert übergeben, die Funktion gibt das Ergebnis zurück. Ist zahl vom Typ float, so ist auch das Ergebnis vom Typ float, ansonsten ist es vom Typ int.

acos()

Die Funktion berechnet den Arcuscosinus im Bogenmaß.

```
float acos(float arg)
```

Ein Wert vom Typ float wird als Argument übergeben.

AddSlashes()

Markiert Sonderzeichen in einem String mit Slashes.

```
String AddSlashes(String arg)
```

Gibt einen String (Zeichenkette) zurück, in dem bestimmten Zeichen ein Backslash »\« vorangestellt wurde. Diese Funktion ist z.B. für Datenbankabfragen wichtig.

add_root()

Methode vom Wurzelobjekt eines XML-Dokuments. Legt ein neues Root-Element an und erschafft so ein neues XML-Dokument.

```
$object->add_root(String name)
```

Die Funktion gibt TRUE oder FALSE zurück.

array()

Erstellt ein Array aus den übergebenen Parametern.

```
array array(...)
```

Mit dem Operator => können diese Parameter identifiziert werden. `array()` ist ein Sprachkonstrukt, das benutzt wird um Arrays zu erzeugen. Es ist keine Funktion.

array_count_values()

Zählt die Häufigkeit des Auftretens aller Elemente und gibt das Ergebnis in einem Array wieder.

```
array array_count_values(array arg)
```

Das Ergebnis ist ein assoziatives Array, das mit dem Namen jedes Elements auf die Häufigkeit zeigt.

array_keys()

Erzeugt ein Array mit allen Schlüsseln eines assoziativen Arrays.

```
array array_keys(array arg)
```

Das Ergebnis ist ein nummerisches Array.

array_merge()

Fasst zwei oder mehr Arrays zu einem neuen Array zusammen.

```
array array_merge(array arg1, array arg2, ...)
```

Sollten die Schlüssel mehrerer assoziativer Arrays übereinstimmen, werden die Werte überschrieben. Das Ergebnisarray wird zurückgegeben.

array_pop()

Gibt das letzte Element eines Arrays zurück und löscht es.

```
mixed array_pop(array arg);
```

Nach der Ausführung der Funktion hat das Array arg ein Element weniger.

array_push()

Hängt einen oder mehrere Werte an ein Array an.

```
int array_push(array array, mixed var, ...)
```

Die Funktion gibt die neue Anzahl der Elemente im Array zurück.

array_reverse()

Dreht die Reihenfolge der Elemente eines Arrays herum.

```
array array_reverse(array arg)
```

Das Ergebnis wird als neues array zurückgegeben.

array_shift()

Entfernt ein Element vom Anfang eines Arrays und gibt es zurück.

```
mixed array_shift(array arg)
```

Nach der Funktion hat das Array ein Element weniger.

array_unshift()

Setzt einen Wert an den Anfang des Arrays.

```
int array_unshift(array array, mixed arg)
```

Die Funktion gibt die neue Anzahl der Elemente des Arrays zurück.

array_values()

Gibt alle Werte eines Arrays in einem Array zurück.

```
array array_values(array arg)
```

Nur sinnvoll bei einem assoziativen Array. Das Gegenstück zu array_leys().

asin()

Berechnet den Arcussinus.

```
float asin(float zahl)
```

Gibt das Ergebnis im Bogenmaß zurück.

atan()

Berechnet den Arcustangens.

```
float atan(float zahl)
```

Gibt das Ergebnis im Bogenmaß zurück.

attributes()

Methode zum Knotenobjekt eines XML-Dokuments.

```
$object -> attributes()
```

Gibt ein Array mit allen Attributen des Elements zurück.

bindec()

Berechnet den Dezimalwert aus einer Binärzahl.

```
int bindec(string arg)
```

Der größte konvertierbare Wert ist ein String aus 31 Einsen, das entspricht 2147483647 dezimal (2 hoch 32 minus 1).

ceil()

Rundet einen Wert mathematisch auf.

```
int ceil(float zahl)
```

Die Funktion gibt eine Ganzzahl zurück, die größer oder gleich dem Parameter ist.

checkdate()

Prüft eine gregorianische Datums- bzw. Zeitangabe auf Gültigkeit.

```
int checkdate(int monat, int tag, int jahr)
```

Gibt TRUE zurück, wenn das Datum gültig ist, ansonsten FALSE. Berücksichtigt auch Schaltjahre.

children()

Methode zu einem Knotenobjekt eines XML-Dokuments.

```
$objekt -> children()
```

Gibt alle Unterknoten des Elements zurück.

chmod()

Ändert die Zugriffsrechte einer Datei auf einem UNIX-System.

```
int chmod(string datei, int rechte)
```

Die Funktion gibt TRUE bei Erfolg zurück.

chown()

Ändert den Eigentümer einer Datei auf einem UNIX-System.

```
int chown(string datei, mixed user)
```

Die Funktion gibt TRUE bei Erfolg zurück.

chr()

Gibt das Zeichen für einen bestimmten ASCII-Code zurück.

```
string chr (int ascii)
```

Der Parameter muss einen Wert zwischen 0 und 255 haben.

class_exits()

Überprüft, ob eine Klasse definiert ist.

```
int class_exists (string name)
```

Wenn die Klasse existiert, gibt die Funktion TRUE zurück.

close()

Schließt eine geöffnete Datei.

```
int close(int filehandle)
```

Gibt bei Erfolg TRUE zurück.

compact()

Setzt Variablen zu einem Array zusammen.

```
array compact(mixed name, ...)
```

Das Ergebnis wird zurückgegeben.

copy()

Kopiert eine Datei auf dem Server.

```
int copy (string quelle, string ziel)
```

Die Funktion gibt bei Erfolg TRUE zurück.

cos()

Berechnet den Cosinus.

```
float cos (float arg)
```

Gibt das Ergebnis im Bogenmaß zurück.

count()

Gibt die Anzahl der Elemente in einem Array zurück.

```
int count (array arg)
```

Gibt 1 zurück, wenn das Argument kein Array ist. Gibt 0 zurück, wenn das Argument nicht existiert.

CURDATE()

MySQL-Funktion: Gibt das aktuelle Datum zurück.

```
date CURDATE()
```

Das Ergebnis ist vom Typ date.

current()

Gibt das aktuelle Element eines Arrays zurück.

```
mixed current (array arg)
```

Ist der Arrayzeiger am Ende des Array, gibt die Funktion FALSE zurück.

CURRENT_TIMESTAMP()

MySQL-Funktion: Gibt das aktuelle Datum und die aktuelle Zeit zurück.

```
date CURRENT_TIMESTAMP()
```

Die Funktion erwartet keine Parameter.

CURTIME()

MySQL-Funktion: Gibt die aktuelle Zeit zurück.

```
time CURTIME()
```

Die Funktion erwartet keine Parameter.

date()

Gibt ein formatiertes Datum oder Zeit zurück.

```
string date (string format, int timestamp)
```

Der Formatierungsstring setzt sich aus unterschiedlichen Steuerzeichen zusammen, die im entsprechenden Kapitel aufgelistet sind.

DATE_ADD()

MySQL-Funktion: Führt eine Datumsberechnung durch.

```
date DATE_ADD(date datum, interval arg)
```

Das angegebene Intervall wird zum Ausgangsdatum dazugerechnet.

DATE_FORMAT()

MySQL-Funktion: Gibt ein formatiertes Datum zurück.

```
string DATE_FORMAT(date arg, string format)
```

Das formatierte String wird vom übergebenen Datum erstellt.

DATE_SUB()

MySQL-Funktion: Führt eine Datumsberechnung durch.

```
date DATE_ADD(date datum, interval arg)
```

Das angegebene Intervall wird vom Ausgangsdatum abgezogen.

decbin()

Wandelt eine Dezimalzahl in eine Binärzahl um.

```
string decbin(int number)
```

Diese Funktion liefert eine binäre Repräsentation von number als String zurück. Der größte konvertierbare Wert für number ist dezimal 2147483647, dies entspricht (2 hoch 32)-1 und liefert einen String aus 31 Einsen.

dechex()

Wandelt eine Dezimalzahl in eine Hexadezimalzahl um.

```
string dechex(int number)
```

Diese Funktion liefert eine hexadezimale Repräsentation von number als String zurück.

decoct()

Wandelt eine Dezimalzahl in eine Oktalzahl um.

```
string decoct(int number)
```

Diese Funktion liefert eine oktale (zur Basis 8) Repräsentation von number als String zurück.

define()

Definiert eine Konstante.

```
int define (string name, mixed value);
```

Der Konstanten-Name wird durch name angegeben. Der Wert wird durch value gesetzt.

die()

Beendet das aktuelle Skript und gibt eine Nachricht aus.

```
void die (string message)
```

Das Spachkonstrukt `die()` gibt eine Nachricht aus und bricht die Skriptausführung ab.

dir()

Gibt ein Objekt auf ein Verzeichnis zurück.

```
new dir(string verzeichnis)
```

Das per `verzeichnis` angegebene Verzeichnis wird geöffnet und ein Objekt darauf zurückgegeben.

domxml_add_root()

Ein neues Root-Element wird für ein XML-Dokument angelegt.

```
int domxml_add_root(object objekt, string name)
```

Die Funktion gibt bei Erfolg TRUE zurück.

domxml_getattr()

Liest ein Attribut eines XML-Elements aus.

```
string domxml_getattr(object objekt, string name)
```

Das Ergebnis wird als String zurückgegeben.

domxml_lastchild()

Gibt das letzte Element eines XML-Knotens zurück.

```
object domxml_lastchild(object objekt)
```

Das Ergebnis ist wieder ein Knoten.

domxml_new_child()

Erstellt ein neues Element für ein XML-Dokument.

```
int domxml_new_child(object obj, string name, string inhalt)
```

Die Funktion gibt bei Erfolg TRUE zurück.

domxml_new_xmldoc()

Legt ein neues XML-Dokument im Speicher an.

```
object domxml_new_xmldoc()
```

Die Funktion gibt ein Objekt zurück.

domxml_parent()

Gibt den übergeordneten Knoten eines Elements zurück.

```
object domxml_parent(object obj)
```

Die Funktion gibt ein Objekt zurück.

domxml_root()

Gibt das Wurzelelement eines XML-Dokument zurück.

```
object domxml_root(object doc)
```

Die Funktion gibt ein Objekt zurück.

domxml_setattr()

Setzt das Attribut eines XML-Elements.

```
int domxml_setattr(object obj, string name, string value);
```

Die Funktion gibt bei Erfolg TRUE zurück.

domxml_set_content()

Legt den Inhalt eines XML-Elements fest.

```
int domxml_set_content(object obj, string content);
```

Die Funktion gibt bei Erfolg TRUE zurück.

dumpmem()

Methode eines Dokumentobjekts. Gibt das XML-Dokument im Speicher als String zurück

```
string $doc -> dumpmem()
```

Für die Ausgabe im Browser muss die Rückgabe mit der Funktion `html-entities()` behandelt werden.

each()

Gibt den nächsten Schlüssel und Wert eines Arrays als Array zurück.

```
array each (array arg)
```

Die Funktion gibt FALSE zurück, falls der Arrayzeiger am Ende des Arrays ist.

empty()

Überprüft, ob eine Variable einen Wert hat.

```
int empty (mixed var)
```

Die Funktion gibt TRUE zurück, wenn die Variable leer oder nicht gesetzt ist.

end()

Setzt den Zeiger auf das letzte Element eines Arrays.

```
void end (array arg)
```

Der Zeiger wird auf das letzte Element gesetzt.

ereg()

Sucht nach Übereinstimmungen mit einem regulären Ausdruck.

```
int ereg (string regexp, string target)
```

Die Funktion gibt TRUE zurück, wenn eine Übereinstimmung vorliegt.

eregi()

Sucht nach Übereinstimmungen mit einem regulären Ausdruck.

```
int eregi (string regexp, string target)
```

Wie `ereg()`, aber ohne Berücksichtigung der Groß- und Kleinschreibung.

ereg_replace()

Sucht nach Übereinstimmungen mit einem regulären Ausdruck und ersetzt ihn durch einen String.

```
string ereg_replace (string regexp, string ersatz, string target)
```

Zurückgegeben wird die geänderte Zeichenfolge, was auch bedeuten könnte, dass die ursprüngliche Zeichenfolge zurückgegeben wird, wenn es keine zu ersetzenden Übereinstimmungen gibt.

eregi_replace()

Sucht nach Übereinstimmungen mit einem regulären Ausdruck und ersetzt ihn durch einen String.

```
string ereg_replace (string regexp, string ersatz, string target)
```

Wie `ereg_replace()`, aber ohne Berücksichtigung der Groß- und Kleinschreibung.

exit()

Beendet die Ausführung des Skripts.

```
void exit()
```

Die Funktion erwartet keinen Parameter.

exp()

Potenziert die Zahl e mit einem weiteren Argument.

```
float exp (float arg)
```

Die Funktion gibt das Ergebnis zurück.

explode()

Zerlegt einen String anhand eines Trennzeichens zu einem Array.

```
array explode (string trenner, string arg)
```

Das Ergebnisarray wird zurückgegeben.

fclose()

Schließt eine geöffnete Datei.

```
int fclose (int handle)
```

Die Funktion gibt TRUE zurück, wenn der Vorgang erfolgreich war.

file()

Liest eine Datei in ein Array ein.

```
array file (string datei)
```

Jede Zeile der Datei entspricht einem Element im Ergebnisarray.

filectime()

Gibt Datum und Uhrzeit der letzten Dateiänderung zurück.

```
int filectime(string datei)
```

Das Ergebnis wird als UNIX-Timestamp zurückgegeben.

fileowner()

Gibt die Benutzer-ID des Dateieigentümers aus.

```
int fileowner(string datei)
```

Wenn ein Fehler auftritt, gibt die Funktion FALSE zurück.

fileperms()

Gibt die Zugriffsrechte einer Datei aus.

```
int fileperms (string datei)
```

Die Funktion gibt FALSE zurück, wenn die Datei nicht existiert.

floor()

Rundet eine Zahl mathematisch ab.

```
int floor (float zahl)
```

Diese Funktion liefert die ganze Zahl, die kleiner oder gleich dem Parameter `number` ist.

fopen()

Öffnet eine Datei der URL.

```
int fopen (string datei, string mode)
```

Gibt ein Handle auf eine Datei oder eine URL wieder.

for()

Sprachkonstrukt: Die `for()`-Schleife gehört zu den komplexen Schleifentypen in PHP.

```
for (expr1; expr2; expr3) Anweisung
```

Der erste Parameter (`expr1`) wird beim Schleifenbeginn geprüft bzw. ausgeführt (ohne jegliche Vorbedingung). Zu Beginn jedes Durchlaufs wird nun `expr2` geprüft. Wenn dieser `TRUE` ist, fährt die Schleife weiter fort mit der Ausführung der nachfolgenden Befehle.

Wenn das Ergebnis FALSE lautet, wird die Schleife beendet. Am Ende jedes Durchlaufs wird nun auch noch expr3 geprüft/ausgeführt.

foreach()

Durchläuft alle Elemente eines Arrays und gibt sie in einer Schleife zurück.

```
foreach (array arg as mixed value) {...}
foreach (array arg as string key => mixed value) {...}
```

Die Funktion kann sowohl für nummerische als auch für assoziative Arrays eingesetzt werden. Achten Sie auf die unterschiedliche Syntax.

fpassthru()

Gibt eine komplette Datei aus.

```
int fpassthru (int handle)
```

Die Datei wird von der letzten Stelle des Dateizeigers ausgegeben. Im Fehlerfall wird FALSE zurückgegeben.

fputs()

Schreibt Daten an die Position des Dateizeigers.

```
int fputs (int handle, string arg)
```

Die Datei gibt bei Erfolg TRUE zurück.

fseek()

Verschiebt den Dateizeiger.

```
int fseek(int steps)
```

Bei einem Fehler wird –1 zurückgegeben, ansonsten 0.

ftp_cdup()

Wechselt auf einem FTP-Server eine Verzeichnisebene höher.

```
int ftp_up (int handle)
```

Gibt bei Erfolg TRUE zurück.

ftp_chdir()

Wechselt das Verzeichnis auf einem FTP-Server.

```
int ftp_chdir (int handle, string verzeichnis)
```

Gibt bei Erfolg TRUE zurück.

ftp_connect()

Stellt eine FTP-Verbindung her.

```
int ftp_connect (string host)
```

Gibt bei Erfolg ein FTP-Handle zurück, ansonsten FALSE.

ftp_delete()

Löscht eine Datei vom FTP-Server.

```
int ftp_delete (int handle, string datei)
```

Die Funktion gibt im Erfolgsfall TRUE zurück.

ftp_get()

Lädt eine Datei vom FTP-Server herunter.

```
int ftp_get (int handle, string local, string remote, int mode)
```

Gibt TRUE oder FALSE zurück. Über den Parameter mode kann zwischen ASCII- und Binärdownload entschieden werden.

ftp_login()

Meldet einen User am FTP-Server an.

```
int ftp_login (int handle, string user, string passwd)
```

Gibt bei Erfolg TRUE zurück.

ftp_mdtm()

Ermittelt den Zeitpunkt der letzten Änderung an einer Datei.

```
int ftp_mdtm (int handle, string datei)
```

Der Zeitpunkt wird als UNIX-Timestamp zurückgegeben. Im Fehlerfall –1.

ftp_mkdir()

Erstellt ein neues Verzeichnis auf einem FTP-Server.

```
int ftp_mkdir (int handle, string verzeichnis)
```

Gibt TRUE oder FALSE zurück.

ftp_nlist()

Gibt eine Verzeichnisübersicht zurück.

```
array ftp_nlist(int handle, string verzeichnis)
```

Das Ergebnis wird als Array zurückgegeben.

ftp_put()

Überträgt eine Datei auf den FTP-Server.

```
int ftp_put (int handle, string remote, string local, int mode)
```

Bei Erfolg wird TRUE zurückgegeben, ansonsten FALSE.

ftp_pwd()

Gibt das aktuelle Verzeichnis auf dem FTP-Server zurück.

```
string ftp_pwd(int handle)
```

Gibt ein Verzeichnis oder FALSE zurück.

ftp_quit()

Schließt eine FTP-Verbindung.

```
int ftp_quit (int handle)
```

Gibt TRUE oder FALSE zurück.

ftp_rawlist()

Gibt eine Verzeichnisübersicht zurück.

```
array ftp_raw(int handle, string verzeichnis)
```

Das Ergebnis wird als Array zurückgegeben (mehr Informationen als `ftp_nlist()`).

ftp_rename()

Ändert den Namen einer Datei auf einem FTP-Server.

```
int ftp_rename (int handle, string name_old, string name_new)
```

Gibt TRUE oder FALSE zurück.

ftp_rmdir()

Entfernt ein Verzeichnis auf dem FTP-Server.

```
int ftp_rmdir (int handle, string verzeichnis)
```

Gibt TRUE oder FALSE zurück.

ftp_size()

Ermittelt die Größe einer Datei auf einem FTP-Server.

```
int ftp_size (int handle, string datei)
```

Gibt die Größe oder –1 zurück.

ftp_systype()

Ermittelt das Betriebssystem des FTP-Servers.

```
string ftp_systype (int handle)
```

Gibt den Systemtyp zurück oder FALSE.

fwrite()

Schreibt Daten in eine geöffnete Datei.

```
int fwrite (int handle, string arg)
```

Gibt TRUE oder FALSE zurück.

getattr()

Methode: Gibt den Wert eines bestimmten Attributs eines XML-Elements zurück.

```
string $object -> getattr(string name)
```

Das Ergebnis wird immer als String zurückgegeben.

getdate()

Gibt Datums- und Zeitangaben zurück.

```
array getdate (int timestamp)
```

Das Ergebnis wird als assoziatives Array zurückgegeben.

getenv()

Gibt die Werte von Umgebungsvariablen wieder.

```
string getenv (string name)
```

Gibt den Wert oder FALSE zurück.

getimagesize()

Gibt die Größe eines Bildes wieder.

```
array getimagesize (string datei)
```

Gibt das Ergebnis in einem Array wieder.

getlastmod()

Gibt das Datum und die Uhrzeit der letzten Änderung am aktuellen Skript zurück.

```
int getlastmod ()
```

Das Ergebnis wird als UNIX-Timestamp zurückgegeben.

getrandmax()

Gibt den größtmöglichen Zufallswert zurück.

```
int getrandmax ()
```

Diese Funktion liefert den höchsten Wert, den ein Aufruf der Funktion rand() auf dieser Plattform zurückgeben kann.

gettimeofday()

Gibt die aktuelle Zeit des Tages zurück.

```
array gettimeofday ()
```

Das Ergebnis wird als Array zurückgegeben.

gettype()

Gibt den Typ einer Variablen zurück.

```
string gettype (mixed var)
```

Der Typ wird als String zurückgegeben. Im Fehlerfall gibt die Funktion FALSE zurück.

get_class()

Gibt den Namen der Klasse eines Objekts zurück.

```
string get_class (object objekt)
```

Der Name wird als String zurückgegeben.

get_class_methods()

Gibt ein Array mit allen Methoden einer Klasse zurück.

```
array get_class_methods (string class)
```

Falls die Klasse nicht existiert, wird FALSE zurückgegeben.

get_meta_tags()

Liest die Metatags einer HTML-Seite aus.

```
array get_meta_tags (string datei)
```

Das Ergebnis wird als Array zurückgegeben.

get_object_vars()

Liefert die Elemente eines Objekts zurück.

```
array get_object_vars (object obj)
```

Das Ergebnis wird als Array zurückgegeben.

get_parent_class()

Gibt den Namen der Oberklasse eines Objekts zurück.

```
string get_parent_class ( object obj)
```

Der Name wird als String zurückgegeben.

gmdate()

Formatiert eine GMT-Zeitangabe.

```
string gmdate (string format, int timestamp)
```

Das formatierte Datum wird als String zurückgegeben.

gmmkdate()

Gibt den UNIX-Timestamp in GMT zurück.

```
int gmmktime (int stunde, int minute, int sekunde, int monat, int
tag, int jahr)
```

Identisch zu mktime(), aber in der GMT-Zeitzone.

header()

Sendet einen HTTP-Header an den Browser.

```
int header (string header)
```

Muss vor der ersten Ausgabe angewendet werden.

hexdec()

Wandelt eine Hexadezimalzahl in eine Dezimalzahl um.

```
int hexdec (string hex)
```

Der Übergabeparameter ist ein String.

htmlentities()

Wandelt Sonderzeichen in HTML-Codes um.

```
string htmlentities (string arg)
```

Gegenwärtig wird der ISO-8859-1-Zeichensatz verwendet.

htmlspecialchars()

Wandelt Sonderzeichen in HTML-Codes um.

```
string htmlentities (string arg)
```

Gegenwärtig wird der ISO-8859-1-Zeichensatz verwendet.

if()

Sprachkonstrukt zur Formulierung von Bedingungen.

```
if (ausdruck) { ... }
```

Wenn der Ausdruck TRUE ist, wird der Bedingungsblock in den geschweiften Klammern ausgeführt.

imagearc()

Zeichnet eine Ellipse.

```
int imagearc (int img, int b, int c, int d, int x, int y, int z, int
col)
```

Zeichnet eine Ellipse um den Mittelpunkt b, c.

imagecolorallocate()

Erstellt eine neue Farbe.

```
int imagecolorallocate (int img, int red, int green, int blue)
```

Erstellt eine Farbe im RGB-System und gibt ein Handle zurück.

imagecolorat()

Ermittelt den Farbwert eines Punktes.

```
int imagecolorat (int img, int x, int y)
```

Gibt den Farbwert zurück.

imagecopy()

Kopiert einen Bildausschnitt.

```
int imagecopy (int zielbild, int quellbild, int ziel_x, int ziel_y,
int quell_x, int quell_y, int quell_hoehe, int quell_breite)
```

Die Funktion erlaubt es, Grafiken zwischen verschiedenen Bildern hin und her zu kopieren.

imagecopyresized()

Kopiert einen Bildausschnitt.

```
int imagecopyresized (int zielbild, int quellbild, int ziel_x, int
ziel_y, int quell_x, int quell_y, int ziel_hoehe, int zie_breite, int
quell_hoehe, int quell_breite)
```

Wie imagecopy(), aber die Funktion erlaubt es, die Bildgröße zu verändern.

imagecreate()

Erzeugt ein neues Bild.

```
int imagecreate (int x, int y)
```

Erzeugt ein Handle, der auf das neue Bild im Speicher zeigt.

imagecreatefromgif()

Erzeugt ein neues Bild aus einer GIF-Datei.

```
int imagecreatefromgif (string datei)
```

Erzeugt ein Handle, der auf das neue Bild im Speicher zeigt.

imagecreatefromjpg()

Erzeugt ein neues Bild aus einer JPG-Datei.

```
int imagecreatefromjpg (string datei)
```

Erzeugt ein Handle, der auf das neue Bild im Speicher zeigt.

imagecreatefrompng()

Erzeugt ein neues Bild aus einer PNG-Datei.

```
int imagecreatefromPNG (string datei)
```

Erzeugt ein Handle, der auf das neue Bild im Speicher zeigt.

imagedashedline()

Zeichnet eine gestrichelte Linie.

```
int imagedashedline (int img, int x1, int y1, int x2, int y2, int
col)
```

Gibt TRUE oder FALSE zurück.

imagedestroy()

Löscht ein Bild aus dem Speicher.

```
int imagedestroy (int img)
```

Gibt TRUE oder FALSE zurück.

imagefill()

Füllt einen Bereich mit Farbe.

```
int imagefill (int img, int x, int y, int col)
```

Füllt den Bereich zwischen zwei Begrenzungen mit Farbe.

imagefilledpolygon()

Zeichnet ein gefülltes Polygon.

```
int imagefilledpolygon (int img, array punkte, int anzahl_punkte,
int color)
```

Die Punkte werden als Array übergeben.

imagefilledrectangle()

Zeichnet ein gefülltes Viereck.

```
int imagefilledrectangle (int img, int x1, int y1, int x2, int y2,
int col)
```

Gibt TRUE oder FALSE zurück.

imagejpeg()

Ausgabe eines Bildes im JPEG-Format.

```
int imagejpeg (int img, [string datei])
```

Der optionale Parameter datei erlaubt es, das Bild in eine Datei zu schreiben.

imageline()

Zeichnet eine Linie.

```
int imageline (int img, int x1, int y1, int x2, int y2, int col)
```

Zeichnet eine gerade durchgezogene Linie ins Zielbild.

imagepng()

Ausgabe eines Bildes im PNG-Format.

```
int imagepng (int img, [string datei])
```

Der optionale Parameter datei erlaubt es, das Bild in eine Datei zu schreiben.

imagepolygon()

Zeichnet ein Vieleck.

```
int imagepolygon (int img, array punkte, int num_punkte, int col)
```

Die Koordinaten der Punkte werden in einem Array übergeben.

imagerectangle()

Zeichnet ein Rechteck.

```
int imagerectangle (int img, int x1, int y1, int x2, int y2, int col)
```

Gibt TRUE oder FALSE zurück.

imagesetpixel()

Zeichnet einen Pixel.

```
int imagesetpixel (int img, int x, int y, int col)
```

Setzt einen Punkt an die angegebenen Koordinaten.

imagestring()

Zeichnet einen String.

```
int imagestring (int img, int font, int x, int y, int string, int
col)
```

Zeichnet einen horizontalen String in das angegebene Bild.

imagestringup()

Zeichnet einen senkrechten String.

```
int imagestring (int img, int font, int x, int y, int string, int col)
```

Zeichnet einen vertikalen String in das angegebene Bild.

imagesx()

Ermittelt die Bildbreite.

```
int imagesx (int img)
```

Gibt den Wert als Integer zurück.

imagesy()

Ermittelt die Bildhöhe.

```
int imagesy (int img)
```

Gibt den Wert als Integer zurück.

imap_8Bit()

Konvertiert einen String in einen quoted-printable kodierten String.

```
string imap_8bit (string arg)
```

Gibt das Ergebnis als String zurück.

imap_base64()

Dekodiert BASE64-codierten Text.

```
string imap_base64 (string arg)
```

Gibt das Ergebnis als String zurück.

imap_binary()

Konvertiert 8Bit-Text in einen BASE64-kodierten String.

```
string imap_binary (string arg)
```

Gibt das Ergebnis als String zurück.

imap_body()

Liest den Körper einer Nachricht.

```
string imap_body (int handle, int num, [int flags])
```

Gibt das Ergebnis als String zurück.

imap_check()

Checkt den Status einer Mailbox.

```
object imap_check (int handle)
```

Gibt ein Objekt mit allen Eigenschaften zurück.

imap_delete()

Markiert eine Nachricht zum Löschen in der Mailbox.

```
int imap_delete (int handle, int num, [int flags])
```

Die Funktion gibt immer TRUE zurück.

imap_expunge()

Löscht alle markierten Nachrichten.

```
int imap_expunge (int handle)
```

Die Funktion gibt immer TRUE zurück.

imap_fetchbody()

Liefert einen bestimmten Abschnitt aus dem body einer Nachricht.

```
string imap_fetchbody (int handle, int num, int part, [int flags])
```

Gibt das Ergebnis als String zurück.

imap_fetchstructure()

Analysiert den Aufbau einer Mail.

```
object imap_fetchstructure (int handle, int num [, int flags])
```

Gibt ein Objekt mit allen Daten zurück.

imap_header()

Gibt den Header einer Nachricht zurück.

```
string imap_header (int handle, int num)
```

Gibt das Ergebnis als String zurück.

imap_headers()

Liefert eine Zusammenfassung von allen Headern der gespeicherten Nachrichten.

```
array imap_headers (int handle)
```

Gibt ein Array mit den Headern aus.

imap_mailboxmsginfo()

Gibt Informationen über das aktuelle Postfach zurück.

```
object imap_mailboxmsginfo (int handle)
```

Das Ergebnis wird als Objekt zurückgegeben.

imap_num_msg()

Gibt die Anzahl der Nachrichten im Postfach zurück.

```
int imap_num_msg (int handle)
```

Das Ergebnis wird als Integer zurückgegeben.

imap_num_recent()

Gibt die Anzahl der neuen Nachrichten im Postfach zurück.

```
int imap_num_recent (int handle)
```

Das Ergebnis wird als Integer zurückgegeben.

imap_open()

Öffnet die Verbindung zu einem Postfach (POP, IMAP, NNTP).

```
int imap_open (string mailbox, string user, string passwd [,int
flags])
```

Die Funktion gibt ein Handle auf die geöffnete Mailbox zurück oder FALSE bei einem Fehler.

imap_ping()

Prüft, ob die Verbindung zu einer Mailbox noch steht.

```
int imap_ping (int handle)
```

Gibt TRUE oder FALSE zurück.

imap_qprint()

Konvertiert einen quoted-printable kodierten String in einen 8bit-String.

```
string imap_qprint (string arg)
```

Gibt das Ergebnis als String zurück.

imap_undelete()

Entfernt eine Löschmarkierung von einer Mail.

```
int imap_undelete (int handle, int num)
```

Gibt TRUE oder FALSE zurück.

imap_utf7_decode()

Dekodiert einen String im modifizierten UTF-7 Format.

```
string imap_utf7_decode (string arg)
```

Gibt das Ergebnis als String zurück.

imap_utf7_encode()

Kodiert Text im modifizierten UTF-7 Format.

```
string imap_utf7_encode (string arg)
```

Gibt das Ergebnis als String zurück.

implode()

Verbindet Arrayelemente zu einem String.

```
string implode (string between, array arg)
```

Über den Parameter between kann ein Verbindungsstring angegeben werden.

include()

Fügt die angegebene Datei in das Skript ein.

```
include (string datei)
```

Kann auch innerhalb von Schleifen verwendet werden.

include_once()

Fügt die angegebene Datei in das Skript ein.

```
include_once (string datei)
```

Kann im Gegensatz zu `include()` nur einmal im Skript verwendet werden.

in_array()

Überprüft, ob ein Element in einem Array vorhanden ist.

```
int in_array (mixed var, array arg)
```

Gibt TRUE oder FALSE zurück.

isset()

Überprüft, ob eine Variable gesetzt ist.

```
int isset (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_array()

Überprüft, ob die Variable ein Array ist.

```
int is_array (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_bool()

Überprüft, ob die Variable ein boolescher Wert ist.

```
int is_bool (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_double()

Überprüft, ob die Variable eine Kommazahl ist.

```
int is_double (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_numeric()

Überprüft, ob die Variable ein nummerischer Wert ist oder ein String.

```
int is_numeric (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_object()

Überprüft, ob die Variable ein Objekt ist.

```
int is_object (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_string()

Überprüft, ob die Variable ein String ist.

```
int is_string (mixed var)
```

Gibt TRUE oder FALSE zurück.

is_subclass_of()

Bestimmt, ob ein Objekt zu einer Subklasse der angegebenen Klasse gehört.

```
int is_subclass_of (object obj, string class)
```

Gibt TRUE oder FALSE zurück.

key()

Gibt den aktuellen Schlüssel eines Arrays zurück.

```
mixed key (array arg)
```

Das Ergebnis wird von der Funktion zurückgegeben.

krsort()

Sortiert ein Array anhand des Schlüssels in umgekehrter Reihenfolge.

```
int krsort (array arg)
```

Die Funktion ist nur bei assoziativen Arrays nützlich.

ksort()

Sortiert ein Array nach dem Schlüssel.

```
int ksort (array arg)
```

Die Funktion ist nur bei assoziativen Arrays nützlich.

lastchild()

Methode eines Knotenobjekts in einem XML-Dokument. Gibt den letzten Unterknoten des ausführenden Knotens zurück.

```
object $objekt -> lastchild ()
```

Die Funktion gibt ein Objekt zurück.

log()

Berechnet den natürlichen Logarithmus.

```
float log (float arg)
```

Diese Funktion liefert den natürlichen Logarithmus (zur Basis e).

ltrim()

Entfernt führende Leerzeichen in einem String.

```
string ltrim (string arg)
```

Das Ergebnis wird zurückgegeben.

mail()

Sendet eine E-Mail.

```
int mail (string to, string subject, string message[, string hea-
ders])
```

Gibt TRUE oder FALSE zurück.

max()

Gibt den größten Wert zurück.

```
mixed max (mixed arg1, mixed arg2,...)
```

Gibt den größten Wert der Übergabeparameter wieder. Ist der erste Wert ein Array, wird der größte Wert im Array bestimmt.

method_exists()

Überprüft, ob eine Methode in der Klasse definiert ist.

```
int method_exists (object obj, string methode)
```

Gibt TRUE oder FALSE zurück.

microtime()

Gibt den UNIX-Zeitstempel in Mikrosekunden wieder.

```
string microtime()
```

Liefert einen String aus zwei Integerwerten zurück (Millisekunden Sekunden).

min()

Gibt den kleinsten Wert zurück.

```
mixed min (mixed arg1, mixed arg2,...)
```

Gibt den kleinsten Wert der Übergabeparameter wieder. Ist der erste Wert ein Array, wird der kleinste Wert im Array bestimmt.

mkdir()

Erstellt ein Verzeichnis auf dem Server.

```
int mkdir (string verzeichnis, int mode)
```

Der Parameter mode bestimmt die Zugriffsrechte für das Verzeichnis.

mktime()

Gibt den UNIX-Timestamp für ein Datum zurück.

```
int mktime (int stunde, int minute, int sekunde, int monat, int tag,
int jahr)
```

Wenn Parameter weggelassen werden, dann ersetzt PHP diese durch die aktuelle Systemzeit.

mt_getrandmax()

Gibt die höchste Zufallszahl zurück.

```
int mt_getrandmax ()
```

Diese Funktion liefert den höchsten Wert, den ein Aufruf der Funktion mt_rand() auf dieser Plattform zurückgeben kann.

mt_rand()

Erzeugt eine Zufallszahl.

```
int mt_rand (int min[, int max])
```

Die erzeugte Zufallszahl ist »besser« als eine Zufallszahl, die mit `rand()` erzeugt wurde.

mt_srand()

Legt den Startwert für die Funktion mt_rand() fest.

```
void mt_srand(int zahl)
```

In der Praxis wird diese Funktion mit der Uhrzeit aufgerufen.

mysql_affected_rows()

Gibt die Anzahl der betroffene Datensätze einer SQL-Anfrage zurück.

```
int mysql_affected_rows (int handle)
```

Das Ergebnis wird als Integerwert zurückgegeben.

mysql_change_user()

Ändert den User eine Datenbankverbindung.

```
int mysql_change_user (string user, string passwd, string db, int
handle)
```

Schlägt der Versuch fehl, bleibt der aktive User bestehen.

mysql_connect()

Baut eine Verbindung zu einer MySQL-Datenbank auf.

```
int mysql_connect (string host, string user, string passwd)
```

Die Funktion gibt ein Handle auf die Verbindung zurück.

mysql_create_db()

Eine neue Datenbank wird erstellt.

```
int mysql_create_db (string name, int handle)
```

Gibt TRUE oder FALSE zurück.

mysql_data_seek()

Bewegt den Datensatzzeiger.

```
int mysql_data_seek (int handle, int num)
```

Gibt TRUE oder FALSE zurück.

mysql_db_name()

Wertet das Ergebnis der Funktion mysql_list_dbs() aus.

```
string mysql_db_name(int erg, int num)
```

Gibt den Namen der Datenbank zurück.

mysql_db_query()

Führt eine SQL-Anfrage aus.

```
int mysql_db_query (string datenbank, string anfrage, int handle)
```

Die Funktion gibt einen Zeiger auf das Ergebnis im Speicher zurück.

mysql_drop_db()

Löscht eine Datenbank.

```
int mysql_drop_db (string datenbank, int handle)
```

Gibt TRUE oder FALSE zurück.

mysql_errno()

Gibt die Fehlernummer des letzten SQL-Fehlers zurück.

```
int mysql_errno (int handle)
```

Sollte im Zusammenhang mit mysql_error() verwendet werden.

mysql_error()

Gibt den Fehlerstring des letzten SQL-Fehlers zurück.

```
string mysql_error (int handle)
```

Sollte im Zusammenhang mit mysql_errno() verwendet werden.

mysql_fetch_array()

Analysiert eine Ergebnisrelation im Speicher.

```
array mysql_fetch_array (int erg[, Typ])
```

Das Ergebnis wird als assoziatives oder nummerisches Array zurückgegeben.

mysql_fetch_assoc()

Analysiert eine Ergebnisrelation im Speicher.

```
array mysql_fetch_assoc (int erg)
```

Das Ergebnis wird als assoziatives Array zurückgegeben.

mysql_fetch_row()

Analysiert eine Ergebnisrelation im Speicher.

```
array mysql_fetch_row (int erg)
```

Das Ergebnis wird als nummerisches Array zurückgegeben.

mysql_field_flags()

Liefert die Optionen einer Spalte.

```
string mysql_field_flags (int erg, int num)
```

Das Ergebnis wird als String zurückgegeben.

mysql_field_name()

Liefert den Namen eines Feldes in einem Anfrageergebnis.

```
string mysql_field_name (int erg, int num)
```

Das Ergebnis wird als String zurückgegeben.

mysql_field_table()

Liefert den Namen der Tabelle eines Feldes zurück.

```
string mysql_field_table (int erg, int num)
```

Das Ergebnis wird als String zurückgegeben.

mysql_field_type()

Liefert den Typ eines Feldes zurück.

```
string mysql_field_type (int erg, int num)
```

Das Ergebnis wird als String zurückgegeben.

mysql_free_result()

Gibt den belegten Speicher einer Ergebnisrelation wieder frei.

```
int mysql_free_results (int erg)
```

Gibt TRUE oder FALSE zurück.

mysql_list_dbs()

Liefert eine Liste der verfügbaren Datenbanken auf dem Server.

```
int mysql_list_dbs (int handle)
```

Gibt einen Zeiger auf das Ergebnis im Speicher zurück.

mysql_list_fields()

Listet die Felder einer Tabelle auf.

```
int mysql_list_fields (string datenbank, string tabelle, int handle)
```

Gibt einen Zeiger auf das Ergebnis im Speicher zurück.

mysql_list_tables()

Listet die Tabellen einer Datenbank auf.

```
int mysql_list_tables (string datenbank, int handle)
```

Gibt einen Zeiger auf das Ergebnis im Speicher zurück.

mysql_num_fields()

Liefert die Anzahl der Felder in einem Ergebnis.

```
int mysql_num_fields (int erg)
```

Das Ergebnis wird als Integerwert zurückgegeben.

mysql_num_rows()

Liefert die Anzahl der Datensätze in einem Ergebnis.

```
int mysql_num_rows (int erg)
```

Das Ergebnis wird als Integerwert zurückgegeben.

mysql_pconnect()

Baut eine persistene Verbindung zu einem MySQL-Datenbankserver auf.

```
int mysql_pconnect (string host, string user, string passwd)
```

Die Funktion gibt ein Handle auf die Verbindung zurück.

mysql_query()

Sendet eine SQL-Anfrage zum Datenbankserver.

```
int mysql_query (string anfrage, int handle)
```

Gibt einen Zeiger auf das Ergebnis im Speicher zurück.

mysql_result()

Liefert ein Ergebnis aus einer Ergebnisrelation im Speicher.

```
string mysql_result (int erg, int index, mixed field)
```

Das Ergebnis wird immer als String zurückgegeben.

mysql_select_db()

Aktiviert eine Datenbank.

```
int mysql_select_db (string database, int handle)
```

Gibt TRUE oder FALSE zurück.

mysql_tablename()

Liefert den Namen einer Tabelle.

```
string mysql_tablename (int erg, int num)
```

Das Ergebnis wird immer als String zurückgegeben.

new_child()

Methode eines Knotenobjekts eines XML-Dokuments: Legt einen Unterknoten an.

```
int $objekt -> new_child (string name, string inhalt)
```

Gibt TRUE oder FALSE zurück.

nl2br()

Wandelt Zeilenumbrüche in den HTML-Tag <BR> um.

```
string nl2br (string arg)
```

Der Ergebnis wird als String zurückgegeben.

next()

Gibt das nächste Element eines Arrays zurück.

```
mixed next (array arg)
```

Der Arrayzeiger wird dabei versetzt.

NOW()

MySQL-Befehl: Gibt die akuelle Uhrzeit und das aktuelle Datum zurück.

```
datetime NOW()
```

Das Ergebnis ist vom Typ datetime.

octdec()

Wandelt eine Oktalzahl in eine Dezimalzahl um.

```
int octdec (string oct)
```

Das Ergebnis wird zurückgegeben.

ord()

Gibt den ASCII-Wert eines Zeichens zurück.

```
int ord (string d)
```

Das Ergebnis ist ein Integerwert zwischen 0 und 255.

parent()

Gibt den übergeordneten Knoten eines Elements zurück.

```
object $object -> parent ()
```

Das Ergebnis ist ein Objekt.

parse_str()

Weist die Werte eines URL-Strings Variablen zu.

```
void parse_str (string arg)
```

Die Variablen werden automatisch gesetzt.

phpinfo()

Erzeugt eine HTML-Ausgabe, mit Informationen über den Server und die genutztenPHP-Version.

```
int phpinfo()
```

Die Ausgabe erstreckt sich über mehrere Browserseiten.

phpversion()

Zeigt die Version des PHP-Interpreters.

```
string phpversion ()
```

Das Ergebnis wird als String zurückgegeben.

PI()

MySQL-Funktion: Gibt den Wert von pi zurück.

```
float PI()
```

Das Ergebnis hat den Wert 3.1415...

pi()

Gibt den Wert von pi zurück.

```
float pi ()
```

Das Ergebnis hat den Wert 3.1415...

prev()

Setzt einen Arrayzeiger ein Element zurück.

```
mixed prev (array arg)
```

Gibt das vorhergehende Element zurück.

print()

Gibt eine String aus.

```
void print (string arg)
```

Der String wird im Browser angezeigt.

printf()

Gibt einen String formatiert aus.

```
int printf (string format, string arg)
```

Der String wird im Browser angezeigt.

RAND()

MySQL-Funktion: Gibt eine Zufallszahl aus.

```
int RAND(int n)
```

Es kann optional der Startwert n angegeben werden.

rand()

Gibt eine Zufallszahl aus.

```
int rand (int min [, int max])
```

Gibt einen Zufallswert zwischen min und max aus.

range()

Gibt ein Array mit fortlaufenden Nummern zurück.

```
array range (int min, int max)
```

Gibt ein Array mit Integerwerten zwischen min und max zurück.

rawurldecode()

Dekodierung von URL-kodierten Strings.

```
string rawurldecode (string arg)
```

Das Ergebnis wird als String zurückgegeben.

rawurlencode()

Codierung von URL-kodierten Strings.

```
string rawurlencode (string arg)
```

Das Ergebnis wird als String zurückgegeben.

read()

Methode eines Verzeichnisobjekts: Gibt den nächsten Eintrag im Verzeichnis zurück.

```
string $object -> read()
```

Das Ergebnis wird als String zurückgegeben.

require()

Der Befehl setzt an seine Stelle den Inhalt der angegebenen Datei (ähnlich dem #include von C).

```
require (string datei)
```

Der Befehl kann nur einmal aufgerufen werden.

reset()

Setzt den internen Zeiger eines Arrays auf das erste Element.

```
mixed reset (array arg)
```

Gibt das erste Element des Arrays zurück.

rewind()

Setzt den Dateizeiger auf das erste Zeichen der Datei.

```
int rewind (int handle)
```

Gibt im Fehlerfall 0 zurück.

root()

Methode eines XML-Dokument-Objekts: Gibt ein Objekt auf das Root-Element zurück.

```
object $objekt -> root()
```

Die Methode erwartet keine Parameter.

round()

Rundet einen Wert auf die nächste Ganzzahl.

```
int round (double value)
```

Das Ergebnis wird zurückgegeben.

rsort()

Sortiert ein Array in umgedrehter Reihenfolge.

```
void rsort(array arg)
```

Die Funktion gibt keinen Wert zurück.

session_decode()

Decodiert die Daten einer Session aus einer Zeichenkette.

```
int session_decode (string data)
```

Die Funktion gibt TRUE oder FALSE zurück.

session_destroy()

Löscht alle Daten einer Session.

```
int session_destroy ()
```

Die Funktion gibt TRUE oder FALSE zurück.

session_encode()

Codiert alle Daten einer Session als Zeichenkette.

```
string session_encode()
```

Die Funktion gibt das Ergebnis als String zurück.

session_id()

Gibt die aktuelle Session-ID zurück.

```
string session_id()
```

Die Funktion gibt das Ergebnis als String zurück.

session_is_registered()

Überprüft, ob eine Variable in der aktuellen Session registriert ist.

```
int session_is_registered (string name)
```

Die Funktion gibt TRUE oder FALSE zurück.

session_name()

Liefert oder setzt den Namen der aktuellen Session.

```
string session_name ()
```

Die Funktion gibt das Ergebnis als String zurück.

session_register()

Registriert eine oder mehrere Variablen in der aktuellen Session.

```
int session_register (mixed var1, mixed var2, ...)
```

Die Funktion gibt TRUE oder FALSE zurück.

session_save_path()

Liefert und/oder setzt den aktuellen Speicherpfad der Session.

```
string session_save_path ()
```

Die Funktion gibt das Ergebnis als String zurück.

session_start()

Initialisiert eine Session.

```
int session_start()
```

Die Funktion gibt TRUE oder FALSE zurück.

session_unregister()

Hebt die Registrierung einer Variablen in der aktuellen Session auf.

```
int session_unregister (string name)
```

Die Funktion gibt TRUE oder FALSE zurück.

session_unset()

Hebt die Registrierung aller Variablen in der aktuellen Session auf.

```
int session_unset ()
```

Die Funktion gibt TRUE oder FALSE zurück.

setattr()

Methode eines Elementobjekts eines XML-Dokuments: Setzt ein Attribut für ein bestimmtes Element.

```
int $object -> setattr(string name, string value);
```

Die Methode gibt TRUE oder FALSE zurück.

setcookie()

Setzt ein Cookie im Browser des Clientrechners.

```
int setcookie (string name, string value, int expire, string path,
string domain, int secure)
```

Die Funktion gibt TRUE bei Erfolg zurück.

settype()

Wandelt den Typ einer Variablen um.

```
int settype (string var, string type)
```

Die Methode gibt TRUE oder FALSE zurück.

set_content()

Methode eines Elementobjekts eines XML-Dokuments. Setzt den Inhalt des aufrufenden Elementobjekts.

```
int $object -> set_content(string content);
```

Die Methode gibt TRUE oder FALSE zurück.

shuffle()

Mischt die Elemente eines Arrays.

```
void shuffle (array arg)
```

Die Funktion gibt keinen Wert zurück.

sin()

Berechnet den Sinus.

```
float sin (float arg)
```

Gibt das Ergebnis im Bogenmaß zurück.

sizeof()

Gibt die Anzahl der Elemente eines Arrays zurück.

```
int sizeof (array arg)
```

Das Ergebnis wird als Integerwert zurückgegeben.

sort()

Sortiert ein Array alphabetisch.

```
void sort (array arg)
```

Die Funktion gibt keinen Wert zurück.

split()

Zerlegt eine Zeichenfolge anhand eines regulären Ausdrucks in ein Array.

```
array split (string muster, string arg)
```

Das Ergebnis wird als Array zurückgegeben.

spliti()

Zerlegt eine Zeichenfolge anhand eines regulären Ausdrucks in ein Array.

```
array spliti (string muster, string arg)
```

Wie `split()`, aber ohne Berücksichtigung der Groß- und Kleinschreibung.

sqrt()

Berechnet die Quadratwurzel.

```
float sqrt (float arg)
```

Das Ergebnis wird zurückgegeben.

srand()

Setzt den Startwert für die Funktion `rand()` fest.

```
void srand (int start)
```

Die Funktion gibt keinen Wert zurück.

strcasecmp()

Vergleich von zwei Zeichenketten ohne Rücksicht auf Groß- und Kleinschreibung.

```
int strcasecmp (string arg1, string arg2)
```

Die Funktion gibt TRUE oder FALSE zurück.

strcmp()

Vergleich von zwei Zeichenketten.

```
int strcmp (string arg1, string arg2)
```

Die Funktion gibt TRUE oder FALSE zurück.

StripSlashes()

Entfernt Slashes zur Codierung von Sonderzeichen.

```
string StripSlashes (string arg)
```

Das Ergebnis wird als String zurückgegeben.

strip_tags()

Entfernt HTML- und PHP-Tags aus einem String.

```
string strip_tags (string arg)
```

Das Ergebnis wird als String zurückgegeben.

strnatcasecmp()

Stringvergleich »natürlicher Ordnung« ohne Unterscheidung der Schreibweise.

```
int strnatcasecmp (string arg1, string arg2)
```

Die Funktion gibt TRUE oder FALSE zurück.

strnatcmp()

Stringvergleich »natürlicher Ordnung« mit Unterscheidung der Schreibweise.

```
int strnatcmp (string arg1, string arg2)
```

Die Funktion gibt TRUE oder FALSE zurück.

strpos()

Ermitteln des ersten Vorkommens innerhalb eines Strings.

```
int strpos (string arg, string muster)
```

Die Position wird als Integerwert zurückgegben.

strrev()

Dreht die Schreibweise eines Strings herum.

```
string strrev (string arg)
```

Das Ergebnis wird als String zurückgegeben.

strrpos()

Ermitteln des letzten Vorkommens innerhalb eines Strings.

```
int strrpos (string arg, string muster)
```

Die Position wird als Integerwert zurückgegeben.

strstr()

Findet das erste Vorkommen eines Strings.

```
int strstr (string arg, string muster)
```

Die Position wird als Integerwert zurückgegeben.

strtolower()

Setzt einen String in Kleinbuchstaben um.

```
string strtolower (string str)
```

Das Ergebnis wird als String zurückgegeben.

strtoupper()

Setzt einen String in Großbuchstaben um.

```
string strtolupper (string str)
```

Das Ergebnis wird als String zurückgegeben.

str_replace()

Ersetzt alle Vorkommen eines Strings in einem anderen String.

```
string str_replace (string muster, string ersatz, string arg)
```

Das Ergebnis wird als String zurückgegeben.

substr()

Gibt einen Teilstring eines Strings zurück.

```
string substr (string arg, string start [,int length])
```

Das Ergebnis wird als String zurückgegeben.

switch()

Sprachkonstrukt: Führt eine Reihe von Vergleichen durch, ähnlich einer if-Bedingung.

```
switch (ausdruck) case ausdruck: ...
```

Der Ausdruck wird mit den darauf folgenden case-Möglichkeiten verglichen. Im Fall von Gleichheit werden die Befehle nach case ausgeführt.

tan()

Berechnet den Tangens.

```
float tan (float arg)
```

Gibt das Ergebnis im Bogenmaß zurück.

time()

Gibt den gegenwärtigen UNIX-Timestamp zurück.

```
int time()
```

Der Timestamp wird als Integerwert zurückgegeben.

trim()

Entfernt Leerzeichen am Anfang und am Ende des Strings.

```
string trim (string arg)
```

Das Ergebnis wird als String zurückgegeben.

ucfirst()

Wandelt das erste Zeichen eines Strings in einen Großbuchstaben um.

```
string ucfirst (string arg)
```

Das Ergebnis wird als String zurückgegeben.

ucwords()

Wandelt das erste Zeichen aller Wörter in einem String in Großbuchstaben um.

```
string ucwords()
```

Das Ergebnis wird als String zurückgegeben.

UNIX_TIMESTAMP()

MySQL-Funktion: Gibt den aktuellen UNIX-Timestamp zurück.

```
int UNIX_TIMESTAMP()
```

Das Ergebnis wird als Integerwert zurückgegeben.

unset()

Löscht eine Variable aus dem Speicher.

```
int unset (mixed var)
```

Die Funktion gibt TRUE oder FALSE zurück.

while()

Schleifenkonstrukt: Führt einen Anweisungblock so lange aus, bis der Ausdruck in den Klammern FALSE ist.

```
while (ausdruck) {...}
```

Keine Funktion, sondern ein Sprachkonstrukt.

wordwrap()

Zeilenumbruch eines Strings an einer angegebenen Stelle unter Verwendung eines angegebenen Trennzeichens.

```
string wordwrap (string str, int width, string break)
```

Das Ergebnis wird als String zurückgegeben.

xmldoc()

Erzeugt ein Objekt mit der DOM-Repräsentation eines XML-Dokumentes.

```
object xmldoc (string datei)
```

Das Ergebnis wird als Objekt zurückgegeben.

xml_error_string()

Gibt die Fehlermeldung des XML-Parsers zurück.

```
string xml_error_string (int code)
```

Das Ergebnis wird als String zurückgegeben.

xml_get_current_byte_index()

Gibt die aktuelle Position des Parsers im Dokument zurück.

```
int xml_get_current_byte_index (int parser)
```

Der Rückgabewert bestimmt die Byteposition im Dokument.

xml_get_current_column_number()

Gibt die aktuelle Position des Parsers im Dokument zurück.

```
int xml_get_current_column_number (int parser)
```

Der Rückgabewert bestimmt den aktuelle Abschnitt im Dokument.

xml_get_current_line_number()

Gibt die aktuelle Position des Parsers im Dokument zurück.

```
int xml_get_current_line_number (int parser)
```

Der Rückgabewert bestimmt die aktuelle Zeile im Dokument.

xml_get_error_code()

Gibt den XML-Fehlercode zurück.

```
int xml_get_error_code(int parser)
```

Der Code kann mit der Funktion xml_get_error_string() übersetzt werden.

xml_parse()

Startet den XML-Parser.

```
int xml_parse (int parser, string data)
```

Gibt TRUE oder FALSE zurück.

xml_parser_create()

Erstellt einen neuen SAX-Parser.

```
int xml_parser_create ([string encoding])
```

Die Funktion gibt ein Handle auf den Parser zurück.

xml_parser_free()

Befreit einen Parser aus dem Speicher.

```
int xml_parser_create (int parser)
```

Gibt TRUE oder FALSE zurück.

xml_parser_get_option()

Gibt die Optionen eines XML-Parsers zurück.

```
mixed xml_parser_get_option (int parser, int option)
```

Das Ergebnis wird von der Funktion zurückgegeben.

xml_parser_set_option()

Setzt die Optionen eines XML-Parsers.

```
int xml_parser_set_option (int parser, int option, mixed value)
```

Gibt TRUE oder FALSE zurück.

xml_set_character_data_handler()

Legt die Funktion für XML-Datenbehandlung fest.

```
int xml_set_character_data_handler (int parser, string funktion)
```

Gibt TRUE oder FALSE zurück.

xml_set_default_handler()

Legt die Funktion für Standardbehandlung von XML-Daten fest.

```
int xml_set_default_handler (int parser, string funktion)
```

Gibt TRUE oder FALSE zurück.

xml_set_element_handler()

Legt die Funktion für die Elementbehandlung eines XML-Dokuments fest.

```
int xml_set_element_handler (int parser, string funktion)
```

Gibt TRUE oder FALSE zurück.

xml_set_processing_instruction_handler()

Legt die Funktion für die Behandlung von PIs in einem XML-Dokument fest.

```
int xml_set_processing_instruction_handler (int parser, string funk-
tion)
```

Gibt TRUE oder FALSE zurück.

Anhang

Im Anhang finden Sie neben den Lösungen zu den Übungen eine Reihe von Informationen, die im Buch selbst keinen Platz gefunden haben. Nutzen Sie vor allem die Linkliste am Ende dieses Kapitels, um im Netz der Netze die aktuellsten Informationen zum Thema zu erhalten.

A.1 Reservierte Schlüsselwörter in PHP4

Folgende Wörter und Begriffe sind von PHP4 reserviert und dürfen nicht für Variablen- oder Funktionsnamen genutzt werden.

and	NULL
E_PARSE	E_ALL
old_function	wakeup
$argv	not
E_ERROR	endwhile
or	__sleep__
as	new
E_WARNING	endswitch
parent	__LINE__
$argc	list()
eval	endif
PHP_OS	endfor
break	virtual()
exit()	include()
$PHP_SELF	enddeclare
case	xor
extends	if
PHP_VERSION	empty()
cfunction	var
FALSE	$HTTP_SERVER_VARS

Tabelle A.1: Reservierte Schlüsselwörter von PHP4

print()	elseif
class	TRUE
for	$HTTP_ENV_VARS
require()	else
continue	$this
foreach	$HTTP_POST_FILES
require_once()	echo()
declare	stdClass
function	$HTTP_POST_VARS
return	die()
default	switch
$HTTP_COOKIE_VARS	$HTTP_GET_VARS
static	do
include_once()	while
endforeach	global
__FILE__	

Tabelle A.1: Reservierte Schlüsselwörter von PHP4 (Forts.)

A.2 Reservierte Schlüsselwörter in MySQL

Folgende Wörter und Begriffe sind von MySQL reserviert und dürfen nicht für Tabellen, Spalten oder Datenbanken genutzt werden.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	AUTO_INCREMENT	BDB
BERKELEYDB	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CASCADE	CASE
CHANGE	CHAR	CHARACTER
COLUMN	COLUMNS	CONSTRAINT
CREATE	CROSS	CURRENT_DATE

Tabelle A.2: Reservierte Schlüsselwörter von MySQL

CURRENT_TIME	CURRENT_TIMESTAMP	DATABASE
DATABASES	DAY_HOUR	DAY_MINUTE
DAY_SECOND	DEC	DECIMAL
DEFAULT	DELAYED	DELETE
DESC	DISTINCTROW	DOUBLE
DESCRIBE	DISTINCT	DROP
ELSE	ENCLOSED	ESCAPED
EXISTS	EXPLAIN	FIELDS
FLOAT	FOREIGN	FROM
FOR	FULLTEXT	FUNCTION
GRANT	GROUP	HAVING
HIGH_PRIORITY	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INNER
INNODB	INSERT	INSERT_ID
INT	INTEGER	INTERVAL
INTO	IS	JOIN
KEY	KEYS	KILL
LAST_INSERT_ID	LEADING	LEFT
LIKE	LIMIT	LINES
LOAD	LOCK	LONG
LONGBLOB	LONGTEXT	LOW_PRIORITY
MASTER_LOG_SEQ	MATCH	MEDIUMBLOB
MASTER_SERVER_ID	MEDIUMINT	MEDIUMTEXT
MIDDLEINT	MINUTE_SECOND	MRG_MYISAM
NATURAL	NOT	NULL
NUMERIC	ON	OPTIMIZE
OPTION	OPTIONALLY	OR
ORDER	OUTER	OUTFILE
OUTFILE	PARTIAL	PRECISION
PRIMARY	PRIVILEGES	PROCEDURE

Tabelle A.2: Reservierte Schlüsselwörter von MySQL (Forts.)

PURGE	READ	REAL
REFERENCES	REGEXP	RENAME
REPLACE	REQUIRE	RESTRICT
RETURNS	REVOKE	RIGHT
RLIKE	SELECT	SET
SHOW	SMALLINT	SONAME
SQL_AUTO_IS_NULL	SQL_BIG_RESULT	TO
SQL_BIG_TABLES	SQL_CALC_FOUND_ROWS	SQL_LOG_BIN
SQL_BUFFER_RESULT	SQL_LOG_OFF	TRAILING
SQL_MAX_JOIN_SIZE	SQL_QUOTE_SHOW_CREATE	SQL_WARNINGS
SQL_SAFE_UPDATES	SQL_SLAVE_SKIP_COUNTER	SSL
SQL_SELECT_LIMIT	SQL_SMALL_RESULT	STARTING
STRAIGHT_JOIN	STRIPED	TABLE
TABLES	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
SQL_LOG_UPDATE	SQL_BIG_SELECTS	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	VALUES	VARBINARY
VARCHAR	VARYING	WHEN
WHERE	WITH	WRITE
YEAR_MONTH	ZEROFILL	

Tabelle A.2: Reservierte Schlüsselwörter von MySQL (Forts.)

A.3 Nutzungsbedingungen von PHP4

Redistribution and use in source and binary forms, with or without modification, is permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name »PHP« must not be used to endorse or promote products derived from this software without prior permission from the PHP Group. This does not apply to add-on libraries or tools that work in conjunction with PHP. In such a case the PHP name may be used to indicate that the product supports PHP.

4. The PHP Group may publish revised and/or new versions of the license from time to time. Each version will be given a distinguishing version number. Once covered code has been published under a particular version of the license, you may always continue to use it under the terms of that version. You may also choose to use such covered code under the terms of any subsequent version of the license published by the PHP Group. No one other than the PHP Group has the right to modify the terms applicable to covered code created under this license.

5. Redistributions of any form whatsoever must retain the following acknowledgement: »This product includes PHP, freely available from *http://www.php.net/*«.

6. The software incorporates the Zend Engine, a product of Zend Technologies, Ltd. (»Zend«). The Zend Engine is licensed to the PHP Association (pursuant to a grant from Zend that can be found at *http://www.php.net/license/ZendGrant/*) for distribution to you under this license agreement, only as a part of PHP. In the event that you separate the Zend Engine (or any portion thereof) from the rest of the software, or modify the Zend Engine, or any portion thereof, your use of the separated or modified Zend Engine software shall not be governed by this license, and instead shall be governed by the license set forth at *http://www.zend.com/license/ZendLicense/*.

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PHP DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CON-

TRACT,STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the PHP Group.

The PHP Group can be contacted via E-Mail at *group@php.net*. For more information on the PHP Group and the PHP project, please see <*http://www.php.net*>.

A.4 Nutzungsbedingungen des Apache Webservers

1. The Apache Software License, Version 1.1, Copyright (c) 2000 The Apache Software Foundation. All rights reserved.
2. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
3. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
4. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
5. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: »This product includes software developed by the Apache Software Foundation (*http://www.apache.org/*).« Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

The names »Apache« and »Apache Software Foundation« must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

Products derived from this software may not be called »Apache«, nor may »Apache« appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <*http://www.apache.org/*>.

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

What is it?

Apache is an HTTP server, originally designed for Unix systems. This is the version of Apache for Microsoft Windows 2000, NT, 98, and 95 systems. Like the Unix version, it includes many frequently requested new features, and has an API that allows it to be extended to meet users' needs more easily. It also allows limited support for ISAPI extensions.

The Latest Version

Details of the latest version can be found on the Apache HTTP server project page under *http://httpd.apache.org/*.

Documentation

The documentation available as of the date of this release is also included, in HTML format, in the htdocs/manual/ directory. For the most up-to-date documentation can be found on *http://httpd.apache.org/docs/*. For Windows specific information, see *http://httpd.apache.org/docs/windows.html*.

WARNING

Apache should never be used as a production server under any consumer operating system such as Windows 95, 98, or ME (Millennium Edition). Only Windows NT 4.0 or 2000 should be considered, and only with appropriate NTFS file system and user security administration. Apache runs on these consumer Windows environments only to provide test, development or trusted intranet server platforms.

Apache on Win32 should be considered initial-release quality code. It has not been subjected to the same stresses on its stability and security that the Unix releases have enjoyed, so there is a greater possibility of undiscovered vulnerabilities to stability or security of the Win32 port.

Apache performs best, and is still most reliable on Unix platforms. Over time the performance, reliability and security for the Apache Win32 port has improved, and continues to improve. Folks doing comparative reviews of webserver performance are still asked to compare against Apache running on a Unix platform such as Solaris, FreeBSD, or Linux.

The Win32 code for Apache 2.0 has been entirely revised and large segments have been rewritten from scratch. Once the Apache 2.0 server is released, we strongly encourage all Win32 users to move to that platform for increased stability and security.

Installation

See the http://httpd.apache.org/docs/windows.html for details of how to install, configure and run Apache. These documents are also accessable from the »Documentation« program group listed within the »Apache Web Server« Start Menu programs group created by the full Apache Win32 program installer.

Known Problems

To get information about the current set of known problems, see the online bug reporting database at *http://www.apache.org/bug_report.html*

Bugs which affect Apache on Windows and not Apache on Unix can be found under the category »os-windows«.

Do not report configuration problems to this database. Please first research the problem you are experiencing on the newsgroup news:comp.infosystems.www.servers.ms-windows and search the bugs database before posting a bug report.

Acknowledgements

We wish to acknowledge the following copyrighted works that make up portions of the Apache software:

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

This package contains software written and copyrighted by Henry Spencer. Please see the file called src/regex/COPYRIGHT.

The NT port was started with code provided to the Apache Group by Ambarish Malpani of ValiCert, Inc. (http://www.valicert.com/).

A.5 GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The »Program«, below, refers to any such program or work, and a »work based on the program« means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term »modification«.) Each licensee is addressed as »you«.

Activities other than copying, distribution and modification are not covered by this license; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this license along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   - You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   - You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this license.
   - If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

     These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in

themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this license.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   - Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   - Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   - Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

5. The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control

compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

6. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

7. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

8. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

9. If, as a consequence of a court judgement or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and

any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

10. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

11. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and »any later version«, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

12. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM »AS IS« WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.6 Lösungen zu den Übungen

Zu Kapitel 3.2.8:

Welchen Sinn haben Kommentare? Auf welche Weise werden sie in PHP realisiert?
Kommentare erlauben es, den Quellcode eines Skripts mit Hinweisen und Bemerkungen zu versehen. Das Skript wird übersichtlicher und ist auch für andere Programmierer lesbar.

Es gibt zwei verschiedene Arten von Kommentaren: einmal den einzeiligen Kommentar, der es erlaubt, einzelne Zeilen auszukommentieren. Die zweite Möglichkeit ist der Kommentar über mehrere Zeilen.

```
//Kommentar
/* Kommentar
zweite Zeile Kommentar */
```

Was sind Anweisungen? Wie werden Anweisungen in PHP markiert?
Eine Anweisung ändert Daten im Programm oder interagiert mit der Programmumgebung. Die Syntax einer Anweisung ist an C angelehnt und endet immer mit einem Semikolon.

Erläutern Sie die Konstanten TRUE und FALSE.
Die Konstanten TRUE und FALSE symbolisieren boolesche Wahrheitswerte. TRUE steht für wahr und FALSE steht für falsch.

Erläutern Sie den Unterschied zwischen einer Zuweisung »by value« und einer Zuweisung »by reference«. Wie sieht dieser Unterschied syntaktisch aus?
Eine Zuweisung »by value« wird immer dann durchgeführt, wenn ein Wert direkt an eine Variable der Funktion übergeben wird. Bei der Zuweisung »by reference« wird nicht der Wert übergeben, sondern die Adresse im Speicher, wo der Wert abgelegt ist. Die Funktion oder Variable greift also auf den Speicher zu.

Werte, die »by reference« übergeben werden, bekommen das kaufmännische »Und« (&) vorangestellt, um sie zu markieren.

Zu Kapitel 3.3.6:

Wozu brauchen wir Operatoren?
Operatoren dienen dazu, Werte miteinander zu verknüpfen. So ist die Interaktion mit der Programmumgebung möglich.

Schreiben Sie ein Programm, das das Quadrat einer Zahl berechnet.

```
<?
$zahl = 5;
$erg = $zahl * $zahl;
echo $erg;
?>
```

Was ist der Unterschied zwischen der Inkrementierung und der Dekrementierung?
Bei der Inkrementierung wird ein Wert erhöht und bei der Dekrementierung verringert.

Erläutern Sie den Unterschied zwischen dem Operator = und dem Operator == .
Der Operator = ist ein Zuweisungsoperator. Er dient beispielsweise dazu, Werte Variablen zuzuweisen. Der Operator == ist ein Vergleichsoperator. Er überprüft, ob zwei Werte den gleichen Wert haben.

Wird or oder and in der Rangfolge zuerst berücksichtigt?
»and« wird vor »or« berücksichtigt.

Zu Kapitel 3.4.5

Was ist an dieser Zuweisung falsch?

```
$kommazahl = 4,5;
```

Da PHP mit der amerikanischen Notation arbeitet, muss anstelle des Kommas ein Punkt gesetzt werden.

Was versteht man unter einer Typumwandlung?
Die Typumwandlung ändert den Typ einer Variablen. Zum Beispiel wird aus einer Kommazahl eine Ganzzahl.

Was gibt dieses Skript aus?

```
<?
$var = "Hallo PHP!";
echo gettype($var);
?>
```

Das Skript gibt den Typ des Strings aus: `String`

Zum Kapitel 3.5.6

Was ist nicht korrekt?

```
$heiß = TRUE;
$kalt = FALSE;
```

Das deutsche Sonderzeichen ß darf nicht in einem Variablennamen verwendet werden.

Was gibt dieses Skript aus?

```
<?
$temperatur = 12;
echo $Temperatur;
?>
```

Das Skript gibt nichts aus, da die Variable `$Temperatur` nicht gesetzt ist.

Erläutern Sie den Unterschied zwischen einer lokalen und einer globalen Variable.
Eine lokale Variable ist nur innerhalb von Funktionen lesbar. Globale Variablen sind im ganzen Skript gültig.

Was ist der Unterschied zwischen dem Vergleichsoperator == und dem Vergleichsoperator === ?
Das doppelte Gleichheitszeichen vergleicht nur die Werte zweier Ausdrücke. Das dreifache Gleichheitszeichen vergleicht auch die Typen.

Was unterscheidet Konstanten von Variablen?
Konstanten können während der Laufzeit eines Programms nicht geändert werden, Variablen schon!

Zu Kapitel 3.6.7

Was ist ein String?
Ein String ist eine Zeichenkette.

Was macht der Punktoperator?
Der Punktoperator verknüpft zwei Strings zu einem neuen.

Welchen Sinn macht eine mächtige Stringbibliothek in PHP?
Da PHP zur Erstellung von dynamischen Webseiten konzipiert wurde, ist es unumgänglich, viel mit Texten zu arbeiten. Darum hat PHP eine große Stringbibliothek.

Was macht die Funktion str_replace()?
Die Funktion ersetzt innerhalb eines Strings bestimmte Suchmuster durch einen Ersatzstring.

Welche Kriterien werden bei Stringvergleichen angelegt?
PHP bewertet bei Stringvergleichen jeden einzelnen Buchstaben in einem String. Die Reihenfolge ist hierbei durch das Alphabet bestimmt, so dass ein a der kleinste und ein z der größte Buchstabe ist. Groß- und Kleinschreibung wird bei zwei der Funktionen ebenfalls berücksichtigt. Großbuchstaben werden vor den Kleinbuchstaben gewertet.

Zum Kapitel 3.7.6

Was ist ein Array?
Ein Array ist eine Liste von unterschiedlichen Werten.

Wie werden Arrays erzeugt? Wie kann man die Elemente ansprechen?
Ein Array wird in PHP über das Schlüsselwort »array« erzeugt. Die einzelnen Elemente werden über ein Offset am Ende des Arraynamens angesprochen.

Erläutern Sie den Unterschied zwischen einem nummerischen und einem assoziativen Array.
In einem nummerischen Array werden die Elemente durch Ganzzahlen angesprochen, beginnend bei 0. Ein assoziatives Array arbeitet mit Schlüsselwörtern.

Was gibt das folgende Listing aus?

```
<?
$liste = array("Brot", "Butter", "Käse", "Kaffee", "Zucker",
"Brot","Kaffee", "Milch", "Zucker", "Brot");
end($liste);
echo current($liste);
?>
```

Das Listing gibt den String »Brot« aus.

Welche Beziehung verknüpft die Funktionen implode() und explode()?
Die Funktion `implode()` wandelt ein Array in einen String um. Die Funktion `explode()` wandelt einen String in ein Array um.

Welche besondere Funktion hat das Array $GLOBALS?
Das Array $GLOBALS speichert alle globalen Variablen in einem Skript.

Was sind mehrdimensionale Arrays und wo finden sie Verwendung?
Ein mehrdimensionales Array besteht aus mehreren verschachtelten Arrays. Es können also so genannte Felder erzeugt werden, die sich in beliebig viele Dimensionen erstrecken. Mehrdimensionale Arrays werden eingesetzt um Datenstrukturen möglichst realitätsnah in einem Programm abbilden zu können.

Zum Kapitel 4.2.8:

Was ist ein Programmblock?
Ein Programmblock wird durch geschweifte Klammern gekennzeichnet und fasst Anweisungen zusammen.

Wann wird ein Bedingungsblock ausgeführt?
Wenn die if-Bedingung TRUE ist.

Wann wird ein else-Block ausgeführt?
Wenn die if-Bedingung FALSE ist.

Was ist der ifelse-Zweig?
Ein bedingter else-Zweig einer if-Anweisung.

Wie werden Bedingungen verknüpft?
Durch logische Operatoren wie AND und OR.

Wofür braucht man die break-Anweisung in der switch-case Konstruktion?
Um mehrere Alternativen bei einem Vergleich anbieten zu können. Im Gegensatz zur einfachen if-Bedingung können mehrere Vergleiche nacheinander durchgeführt werden.

Zum Kapitel 4.3.7:

Was gibt dieses Programm aus?

```
<?
$i = 10;
while($i > 0)
{
       echo "i hat den Wert $i !<br>";
       $i--;
}
?>
```

Das Programm gibt zehn Zeilen aus, die den Wert $i bis 1 herunterzählen.

Was ist der Unterschied zwischen der while()-Schleife und der do-while()-Schleife?
Die while()-Schleife überprüft die Bedingung zu Beginn des Anweisungsblocks. Die do-while()-Schleife überprüft die Bedingung, nachdem die Anweisungen schon ausgeführt wurden. D.h. einmal wird der Block auf jeden Fall ausgeführt.

Was sind Endlosschleifen?
Schleifen, die niemals die Abbruchbedingung erfüllen und aufgrund dessen nicht enden.

Was macht die for()-Schleife so flexibel?
In der for()-Schleife können sowohl die Zählvariablen als auch die Abbruchbedingungen sowie Veränderungen der Zählvariablen im Kopf der Schleife definiert werden.

Welchen Sinn hat die foreach()-Schleife?
Mit der foreach()-Schleife können alle Elemente eines Array der Reihe nach bearbeitet werden.

Erläutern Sie den Unterschied zwischen break und continue!
break beendet eine Schleife ohne Rücksicht auf den momentanen Status. Das Programm wird in der Zeile nach dem Ende der Schleife fortgesetzt. continue bricht den aktuellen Durchlauf der Schleife ab und beginnt die Ausführung von vorne.

Zum Kapitel 4.4.4:

Welche Vorteile hat der Einsatz von Funktionen?
Funktionen erlauben es, Gruppen von Anweisungen zusammenzufassen und über einen Funktionsnamen aufzurufen. So ist es möglich, immer wiederkehrende Anweisungen zentral zu definieren und bei Bedarf aufzurufen.

Was sind rekursive Funktionen?
Rekursive Funktionen rufen sich selbst auf, um eine Problematik zu lösen.

Was ist der Unterschied zwischen include und require?
Der Befehl `require()` wird vom PHP-Interpreter vor der eigentlichen Ausführung des Skriptes bearbeitet und das auch nur ein einziges Mal. Im Gegensatz dazu wird die Funktion `include()` erst während der Laufzeit des Programms interpretiert.

Zum Kapitel 5.2.4:

Wie wird die Konstante pi in PHP verwaltet?
Als Funktion `pi()`.

Auf welchem Wert basieren die Zufallszahlen in PHP?
Auf der aktuellen Systemzeit.

Welche Zahlensysteme unterstützt PHP?
Dezimalsystem, Hexadezimalsystem, Binärzahlen, Oktalzahlen.

Zum Kapitel 5.3.5:

Was ist der UNIX-Timestamp?
Der UNIX-Timestamp gibt die Anzahl aller Sekunden seit dem 1.1.1970 an.

Was gibt dieses Skript aus?

```
<?
echo date("d.M Y H:i:s");
echo "<br>";
?>
```

Das aktuelle Datum und die Uhrzeit.

Was gibt die Funktion getdate() zurück?
Das aktuelle Datum und die Uhrzeit in einem assoziativen Array.

Zum Kapitel 5.4.1:

Welchen Sinn haben PHP-Informationsfunktionen?
Sie erlauben es, Informationen über die verwendete PHP-Distribution zu erhalten. Das kann nützlich sein, wenn es darum geht, vorhandene Skripte an den Server anzupassen.

Was macht die Funktion phpinfo()?
`phpinfo()` gibt einen ausführlichen Überblick über den Server sowie die PHP-Distribution.

Wie würden Sie feststellen, wann Sie zuletzt an einem Skript gearbeitet haben?
Mit der Funktion `getlastmod()`, die den genauen Zeitpunkt der letzten Modifikation zurückgibt.

Zum Kapitel 6.4.1:

Nach welchem System arbeitet die PHP-Fehlersystematik?
PHP unterscheidet zwischen vier Klassen von Ereignissen, die in einem Programm auftreten können. Jedem Ereignis wird ein so genannter Bitwert zugeordnet, der es erlaubt, den Ereignistyp festzulegen.

Was ist das Problem der Zeilenangabe bei Fehlermeldungen?
Der PHP-Interpreter sucht fehlende Zeichen so lange, bis er entweder am Ende des Programms ist oder es syntaktisch keine andere Möglichkeit mehr gibt. An dieser Stelle wird dann erst die Fehlerausgabe gemacht, oft leider viel zu spät.

Was sind semantische Fehler?
Inhaltliche Fehler in einem syntaktisch korrekten Programm.

Welche Möglichkeiten gibt es, semantische Fehler zu finden?
Die einzige Chance, semantische Fehler zu finden, besteht darin, die Zeilen des »verdächtigen« Skripts per Hand nachzuvollziehen. Es erweist sich oft als hilfreich, die betroffenen Variablen während der Laufzeit zu überprüfen oder auszugeben.

Was sind Laufzeitfehler?
Laufzeitfehler können auftreten, wenn während der Abarbeitung eines Skripts Zugriffe auf fremde Ressourcen, wie zum Beispiel Dateien oder Datenbanken, durchgeführt werden.

Wie kann man Laufzeitfehlern begegnen?
PHP bietet die Möglichkeit ein Skript sauber zu beenden und eine Meldung im Browser auszugeben. Die Funktionen hierzu lauten `exit` und `die()`.

Zum Kapitel 7.3.11

Was ist der Unterschied zwischen GET und POST?
GET übergibt Variablen über die URL der Webseite. POST übergibt die Werte als Umgebungsvariablen an das Skript.

Was ist der Unterschied zwischen Checkboxen und Radiobuttons?
Zusammengehörende Checkboxen können gleichzeitig aktiviert sein, Radiobuttons nicht. Sie werden zur Auswahl von Alternativen eingesetzt.

Wie wird das Ziel einer Formularanfrage festgelegt?
Über das Attribut ACTION.

Welche Variablen übergibt dieses Formular?

```
<FORM ACTION="pwd.php" METHOD="post">
Name:<BR>
<INPUT TYPE="text" NAME="name"><BR>
Passwort:<BR>
<INPUT TYPE="password" NAME="pass">
<BR><INPUT TYPE="submit" VALUE="abschicken">
</FORM>
```

Die Variablen $name und $pass.

Welche Möglichkeiten gibt es noch, um Übergabewerte auszulesen?
Die globalen Arrays $HTTP_GET_VARS[] und HTTP_POST_VARS[].

Welche Schlüssel und welche Werte übergibt dieser Link?

```
<a href="go.php?la=de&site=home&id=7462">klick mich!</a>
```

Die Schlüssel la, site und id mit den Werten »de«, »home« und 7426.

Was ist ein DAU? Was macht ein DAU?
Ein DAU ist der Dümmste Anzunehmende User. Ein DAU macht alles falsch, was man falsch machen kann.

Welche Vorteile können dynamische Formulare haben?
Ein dynamisches Formular kann automatisch an die Bedürfnisse eines User angepasst werden.

Zum Kapitel 7.4.8:

Was sind Cookies und welchen Zweck haben sie?
Cookies dienen dazu, Informationen lokal auf dem Rechner des Users abzulegen.

Warum haben Cookies diesen Namen?
»For no special reason!«

Wie löscht man Cookies manuell?
Indem man sie mit einem Datum in der Vergangenheit überschreibt.

Warum könnte ein User Cookies ablehnen?
Um zu verhindern, dass ein Skript Daten über ihn sammelt und beispielsweise zu Werbezwecken verwendet.

Zum Kapitel 7.5.4:

Was sind Sessions?
Sessions sind seitenübergreifende Sitzungen eines Users.

Welchen Zweck haben Sessions?
Sessions erlauben es, Daten über einen User bequem über die Lebenszeit eines Skripts hinaus zu speichern.

Was ist eine Session-ID?
Die Session-ID identifiziert einen einzelnen User und ordnet ihm seine gespeicherten Daten zu.

Warum muss in diesem Skript die Session nicht gestartet werden?

```
<?
$var = "Hallo PHP!";
session_register("var");
echo $var;
?>
```

Weil über den Befehl `session_register()` implizit eine Session gestartet wurde.

Wie kann die Session-ID übergeben werden?
Über alle Möglichkeiten, die das HTTP-Protokoll bietet.

Warum sollte man eine Session immer auf Gültigkeit überprüfen?
Um abgelaufene Sessions zu identifizieren und so Sicherheitslücken zu schließen.

Zum Kapitel 8.2.3:

Was ist der Unterschied zwischen HTML-Text und normalem Fließtext?
HTML-Text enthält Formatierungen in Form von HTML-Tags.

Warum werden Sonderzeichen in HTML codiert?
Damit die Webseite landesübergreifend kompatibel ist und korrekt dargestellt wird.

Warum werden Sonderzeichen in einer URL codiert?
Damit die Daten korrekt übergeben werden.

Zum Kapitel 8.4.3:

Was sind reguläre Ausdrücke?
Ein regulärer Ausdruck (engl.: Regular Expression) ist eine Art Mustervorlage, die es erlaubt Übereinstimmungen in Zeichenketten zu finden.

Was ist der Unterschied zwischen Zeichen und Zeichengruppen?
Eine Zeichengruppe ist die Zusammenfassung von unterschiedlichen Zeichen, die alternativ für einander stehen können.

Erklären Sie den Unterschied zwischen den regulären Ausdrücken A|B|C und [ABC]!
Es gibt keinen.

Welchen Bereich deckt die folgende Zeichengruppe ab: [A-No-z]?
Alle Großbuchstaben von A bis n und alle Kleinbuchstaben von o bis z.

Was sind Wildcards?
Jokerzeichen für ein beliebiges Zeichen.

Warum gibt das folgende Skript FALSE aus?

```
<?
$str = "PHP ist toll!";
$bool = ereg("PHP$", $str);
echo $bool;
?>
```

Weil der String »PHP« nicht am Ende der Zeichenkette steht.

Was ist der Unterschied zwischen ereg() und eregi() ?
Die Funktion `eregi()` berücksichtig keine Groß- und Kleinschreibung.

Zum Kapitel 9.3.7:

Was unterscheidet die objektorientierte Programmierung von anderen Theorien?
Die OOP arbeitet mit Objekten. Ein Objekt ist eine Zusammenstellung aus Eigenschaften und Methoden, die die Fähigkeiten des Objekts darstellen. In der OOP gibt es keine regulären Funktionen oder Variablen, sondern nur Methoden und Eigenschaften.

Was ist der Unterschied zwischen einer Methode und einer Funktion?
Eine Funktion kann unabhängig in einem Programm verwendet werden. Eine Methode kann nur über ein Objekt ausgeführt werden und bezieht sich auch auf dieses.

Was wird zwischen Klassen vererbt?
Es können Eigenschaften und Methoden vererbt werden.

Was unterscheidet Objekte von Klassen?
Ein Objekt wird aus einer Klasse abgeleitet, d.h. es konkretisiert das Modell, das in der Klasse angelegt wurde. Es kann mehrere Objekte eines Typs geben, aber immer nur eine Klasse.

Welche Funktion hat das Schlüsselwort extends?
Es leitet die Vererbung in einer Klassendeklaration ein.

Welchen Sinn macht ein Konstruktor?
Ein Konstruktor wird immer dann aufgerufen, wenn ein neues Objekt aus einer Klasse erstellt wird. Er kann dazu dienen, Initialisierungsmaßnahmen zu treffen oder weitere Objekt zu erschaffen, die für das Objekt notwendig sind.

Zum Kapitel 10.7:

Was ist der Unterschied zwischen den Zugriffsmodi w+ und a+?
Der Zugriffsmodus w+ erlaubt den Schreib- und Lesezugriff auf eine Datei. Bevor die Datei geöffnet wird, wird der gesamte Inhalt gelöscht. Mit a+ wird ebenfalls ein Schreib- und Lesezugriff geöffnet, ohne allerdings die Datei vorher zu leeren.

Kann PHP auf Dateien anderer Server zugreifen?
Ja, PHP kann über das HTTP-Protokoll auf andere Dateien zugreifen. Allerdings nur im Lesezugriff.

Welchen Sinn macht ein Dateizeiger?
Der Dateizeiger markiert die Stelle in der Datei, die zuletzt ausgelesen/geschrieben wurde. Er wird automatisch durch Lese-/und Schreibzugriffe versetzt.

Was macht das Sonderzeichnen \n ?
Es steht für einen Zeilenumbruch.

Was gibt die Funktion dir() zurück?
Ein Handle auf ein Verzeichnis.

Gibt es einen Verzeichniszeiger?
Ja, er zeigt auf die zuletzt zurückgegebene Datei.

Zum Kapitel 11.3.7:

Erklären Sie den Unterschied zwischen dem externen und dem internen Schema einer Datenbank!
Das interne Schema beschreibt eine Datenbank auf der physikalischen Ebene. Es legt fest, wie und wo die Daten auf dem Rechner gespeichert werden. Das externe Schema beschreibt eine Datenbank aus der Sicht der Benutzerschnittstellen. Es gibt immer mehrere externe Betrachtungsweisen, da eine Datenbank von vielen unterschiedlichen Applikationen genutzt wird.

Was ist eine Relation?
Eine Relation ist eine Tabelle, die aus unterschiedlichen Datensätzen besteht.

Was ist der Unterschied zwischen SQL und MySQL?
SQL ist eine Abfragesprache für relationale Datenbanken. MySQL ist eine Datenbanksoftware.

Wofür steht der Ausdruck NULL?
Keine Daten.

Was ist der Unterschied zwischen CHAR und VARCHAR?
CHAR beschreibt einen Datentyp von fester Länge. VARCHAR beschreibt einen Datentyp von variabler Länge.

Was ist ein Schlüssel?
Ein Schlüssel ist eine bestimmte Spalte einer Relation, die einen Datensatz eindeutig identifiziert.

Was ist der Unterschied zwischen UPDATE und REPLACE?
UPDATE ändert einen bestehenden Datensatz. REPLACE ersetzt einen bestehenden Datensatz.

Was ist falsch an dieser Anweisung?

```
SELECT * FROM telnum WHERE name == "Dirk";
```

Der Vergleichsoperator für Werte lautet in SQL = und nicht == wie in PHP.

Können mathematische Funktionen auch in anderen nicht-MySQL-Datenbanken angewendet werden?
Nein, alle mathematischen Funktionen sind nur in der MySQL-Datenbank einsetzbar.

Was unterscheidet Aggregatfunktionen von anderen Funktionen?
Ein Aggregatfunktion bezieht sich nicht auf einen einzelnen Datensatz, sondern eine komplette Spalte.

Zum Kapitel 11.6:

Was ist der Unterschied zwischen den Funktionen mysql_connect() und mysql_pconnect()?
Die Funktion `mysql_pconnect()` erschafft eine persistene Verbindung zu einer Datenbank. Die Funktion `mysql_connect()` nicht.

Was bedeutet deprecated?
Es bedeutet veraltet. Die Funktion oder Methode sollte nicht mehr verwendet werden.

Was ist eine Ergebnisrelation?
Eine Tabelle im Speicher, die das Ergebnis einer SQL-Anfrage symbolisiert.

Gibt die Funktion mysql_fetch_array() ein nummerisches oder ein assoziatives Array zurück?
Die Funktion kann beides zurückgegeben.

Was macht die Funktion mysql_list_fields() ?
Sie gibt eine Ergebnisrelation zurück, die Informationen über eine Tabelle speichert.

Was ist der Unterschied zwischen den Funktionen mysql_error() und mysql_errno?
Die Funktion `mysql_errno()` gibt die aktuelle Fehlernummer zurück, die Funktion `mysql_error()`den aktuellen Fehlerstring.

Warum sollten Sie ein Konfigurationsskript benutzen?
Um alle Zugangsdaten zentral verwalten zu können.

Zum Kapitel 12.6:

Was ist die Voraussetzung für die Erstellung von dynamischen Grafiken?
Das Grafikmodul GD.

Was gibt die Funktion imagecreate() zurück?
Ein Handle auf eine Grafik im Speicher.

Zeichnet die Funktion imagearc() Kreise oder Ellipsen?
Beides, je nachdem welche Parameter man übergibt. Streng genommen ist jeder Kreis eine Ellipse.

Was unterscheidet die Funktion imagerectangle() von der Funktion imagefilledrectangle()?
Die Funktion imagerectangle() zeichnet ein Rechteck. Die Funktion imagefilledrectangle() zeichnet ein gefülltes Rechteck.

Wie können dynamische Grafiken in eine HTML-Datei gebracht werden?
Über den <IMG...>-Tag.

Zum 13.6 Kapitel:

Was bedeutet FTP und was unterscheidet dieses Protokoll vom HTTP-Protokoll?
FTP steht für File Transfer Protokoll und erlaubt im Gegensatz zum HTTP-Protokoll auch den Schreibzugriff auf andere Server.

Warum muss sich ein User für einen FTP-Transfer identifizieren?
Damit der entfernte Server den User verifizieren kann und ihm die entsprechenden Rechte einräumt.

Warum bietet FTP die Möglichkeit auf dem Server zu navigieren?
Um das Verzeichnissystem zu erforschen und Zieldateien aus einem anderen Directory zu erreichen.

Was ist der Unterschied zwischen den Funktionen ftp_chdir() und ftp_cdup()?
Die Funktion `ftp_chdir()` wechselt in ein beliebiges Verzeichnis. Die Funktion ftp_cdup() wechselt eine Verzeichnisebene höher.

Was ist der Unterschied zwischen den Funktionen ftp_rawlist() und ftp_nlist()?
Beide Funktion geben eine Verzeichnisübersicht zurück. Die Funktion `ftp_rawlist()` gibt im Vergleich aber mehr Informationen aus.

Was ist bei der Funktion ftp_rmdir() zu beachten?
Das Verzeichnis muss komplett leer sein, ehe es gelöscht werden kann.

Zum Kapitel 14.6:

Was bedeutet SMTP und wo wird dieses Protokoll eingesetzt?
Es bedeutet Send Mail Transfer Protokoll und wird zum Versenden von E-Mails verwendet.

Wo liegt der Unterschied zwischen POP3 und IMAP?
Beide Protokolle dienen dazu, E-Mails von einem Server abzurufen. IMAP bietet allerdings eine viel größere Vielfalt von Möglichkeiten.

Über welche Header-Bezeichnung wird die Dringlichkeitsstufe festgelegt?
Über die Header-Angaben X-Priority und MSMail-Priority werden Angaben zur Dringlichkeit einer Mail gemacht.

Wo ist der Fehler im folgenden Programm?

```
<?
$mbox = imap_open ("{mail.server.de/pop3:143}INBOX", "user",
"passwd");
if ($mbox) {echo "Verbindung erfolgreich geöffnet!";
imap_close($mbox);
?>
```

Das POP3-Protokoll verwendet den Port 110, im Skript ist der Port 143 angegeben worden.

Welche Informationen hat der Header einer E-Mail?
Er gibt an, aus welchen Teilen die Mail besteht und wie sie codiert sind.

Was ist der Unterschied zwischen den Funktionen imap_header() und imap_headers() ?
Die Funktion `imap_header()` liest den Kopf einer Nachricht aus, während die Funktion `imap_headers()` die Informationen aller Nachrichten ausliest.

Welche Aufgabe hat der MIME-Type einer Mail?
Er bestimmt den Typ der Mail.

Kann eine Mail mehrere Header haben?
Ja, wenn sie aus mehreren Teilen besteht.

Wie werden Bilder für den Mailversand codiert?
Sie werden im base64-Format codiert.

Warum werden Bilder für den Mailversand codiert?
Die SMTP-Server unterstützen nur die 7Bit-ASCII-Verschlüsselung. Um einen Mailanhang zu versenden, ist es deshalb notwendig, diesen auf das entsprechende Zeichenformat zu reduzieren.

Zu Kapitel 15.5.5:

Was ist der Unterschied zwischen XML und HTML?
HTML ist ein Subset von XML und richtet sich nach einer bestehenden DTD.

Welchen Vorteil bietet XML?
XML bietet die Möglichkeit, eigene Tags zu definieren und so eine eigene Auszeichnungssprache zu entwickeln. Daten können in einem systemübergreifenden Format formatiert und interpretiert werden.

Was ist der Unterschied zwischen einem Element und einem Attribut?
Attribute dienen dazu, Elemente mit weiteren Informationen zu versehen.

Was sind PIs?
PIs sind Processing Instructions. Sie erlauben es, innerhalb eines XML-Dokuments Informationen zur Verarbeitung zu hinterlegen.

Warum wird der folgende XML-Code von jedem Parser ignoriert?

```
<!-- <test>Das ist ein Test!/<test> -->
```

Weil er in Kommentarzeichen gesetzt ist.

Welche Aufgabe hat eine DTD?
Die Document Type Definition ist ein Regelwerk, das festlegt, wie ein XML-Dokument aufgebaut werden soll.

Was unterscheidet konkrete Daten von abstrakten Einheiten?
Abstrakte Einhheiten sind Formatierungselemente in einem XML-Dokument, welche die konkreten Daten in eine lesbare Form bringen.

Erklären Sie den Unterschied zwischen wohlgeformten und gültigen Dokumenten.
Ein wohlgeformtes Element richtet sich nach den elementaren Regeln der XML-Syntax. Ein gültiges Dokument entspricht einer vorgegebenen DTD.

Wann sollte eine externe DTD einer internen vorgezogen werden?
Eine externe DTD erlaubt es, die Formatierungsanweisungen außerhalb von Dokumenten zu speichern. So ist es möglich, dass sich mehrere Dokumente an einer DTD-Datei orientieren. Außerdem wird das XML-Dokument übersichtlicher.

Erklären Sie das SAX-Modell!
Das SAX-Modell basiert auf der ereignisgestützten Interpretation eines XML-Dokuments. Jede Datenstück ruft eine Funktion auf, die für die Verarbeitung dieser Information geschrieben werden muss.

Zum Kapitel 15.6.6

Erklären Sie den Unterschied zwischen dem SAX-Parser und dem DOM-Parser!
Im Gegensatz zum SAX-Parser (siehe letzte Frage) sieht der DOM-Parser das XML-Dokument als einen Informationsbaum, der sich mit jedem weiteren Element öffnet. Über so genannte Knoten werden die Informationen identifiziert und ausgelesen.

Warum ist es unter Umständen besser, das SAX-Modell zu verwenden?
Das DOM-Modell kopiert das komplette XML-Dokument in den Speicher, um alle Informationen jederzeit präsent zu haben. Besonders bei großen Dokumenten kann das zu Performanceverlusten führen.

Was gibt die Dokumenteigenschaft »standalone« zurück?
Ob das Dokument auf weitere Daten zurückgreift oder ob es allein steht.

Was unterscheidet die DOM-Methoden von den DOM-Funktionen?
Jede DOM-Methode hat in der Regel ein nicht-objektorientiertes Gegenstück als Funktion. Technisch gesehen gibt es ansonsten keinen Unterschied.

Was ist ein Knoten?
Ein Knoten repräsentiert ein Datenelement im XML-Baum. Über Knoten werden Informationen identifiziert und im Baum navigiert.

Was gibt die Methode children() zurück?
Ein Array mit Objekten, die alle Kindknoten des aufrufenden Objekts speichern.

Was gibt die Methode dumpmem() zurück?
Den kompletten XML-Baum, wie er im Speicher abgebildet ist.

A.7 Liste interessanter Links ins WWW

www.php.net

Die englische Seite php.net ist die bekannteste Entwicklerseite zum Thema PHP. Sie finden hier alle Informationen rund um PHP. Wenn Sie eine Frage zu PHP haben, ist dies die erste Adresse im Netz der Netze.

www.zend.com

Zend gehören zu den Entwicklern von PHP 4 und haben zum Ziel, PHP als den großen Standard in der Webentwicklung zu etablieren. Neben aktuellen Informationen und einer Befehlsreferenz und Beispielskripten finden Sie hier auch alle Neuigkeiten um PHP und immer die aktuellste Version der Skriptsprache.

www.mysql.com

Die Seite im Netz zur Open Source Distribution MySQL.

www.apache.org

Die Apache Group im Netz. Hier finden Sie alle Projekte und kostenlose Software von Apache.

www.phpbuilder.com

Englische Seite für Webentwickler mit PHP. Hier finden Sie Tutorials und jede Menge Informationen zum Thema PHP.

www.phpwizard.net/projects/phpMyAdmin

Hier können Sie die bekannte kostenlose PHP-Software herunterladen, die zur einfachen Verwaltung einer MySQL-Datenbank geschaffen wurde. Der Standard in der Webprogrammierung auf Datenbankbasis.

www.php-center.de

Deutsche Entwicklerseite zum Thema PHP. Hier finden Sie Diskussionsforen, kostenlose Skripte und Neuigkeiten rund um PHP.

kulturbrand.de

Die Seite des Autors. Hier bekommen Sie Updates zum Buch und haben die Möglichkeit mir Feedback zu diesem und anderen Themen zu geben.

Stichwortverzeichnis

C

D

E

F

G

T

U

V

W

X

Y

Z