open source library

Florence Maurice

PHP 5.3 + MySQL 5.1

Der Einstieg in die Programmierung dynamischer Websites

ADDISON-WESLEY

PHP 5.3 + MySQL 5.1

Florence Maurice

open source library

PHP 5.3 + MySQL 5.1

Der Einstieg in die Programmierung dynamischer Websites

ADDISON-WESLEY

An imprint of Pearson Education

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

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

10 9 8 7 6 5 4 3 2 1

12 11 10

ISBN 978-3-8273-2723-9

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

Einbandgestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Lektorat: Boris Karnikowski, bkarnikowski@pearson.de
Fachlektorat: Hagen Graf, cocoate.com
Korrektorat: Brigitte Hamerski, Willich
Herstellung: Monika Weiher, mweiher@pearson.de
Satz: Reemers Publishing Services GmbH, Krefeld (www.reemers.de)
Druck: Bosch Druck, Ergolding
Printed in Germany

Inhaltsübersicht

Inhaltsverzeichnis

Vorwort

PHP ist eine äußerst beliebte Skriptsprache zur serverseitigen Programmierung. Mit PHP können so genannte dynamische Seiten erstellt werden. Das sind Seiten, die jedes Mal, wenn sie aufgerufen werden, neu, d. h. meist mit aktuellen Daten, erzeugt werden. Besonders beliebt ist PHP in der Kombination mit dem Datenbanksystem MySQL, da beide frei zur Verfügung stehen. Mit PHP können Blogs, Content Managementsysteme, Shop-Systeme, Foren, Bildergalerien usw. usf. programmiert werden.

Die Abkürzung PHP selbst steht für PHP Hypertext Preprocessor.

PHP hat viele Vorteile:

- PHP ist speziell für dynamische Webseiten entwickelt worden – das bedeutet, alle Funktionen sind genau darauf zugeschnitten.
- Es ist relativ einfach zu erlernen, ...
- ... trotzdem ausgereift: PHP liegt derzeit in Version 5.3 vor.
- PHP kann sowohl prozedural als auch objektorientiert programmiert werden und ist damit auch für den Einsatz bei größeren Projekten geeignet.
- PHP ist eine äußerst mächtige Skriptsprache. Mit PHP können Sie mit Datenbanken oder auch mit Dateien arbeiten, aber Sie können auch Bilder bearbeiten – beispielsweise Vorschaubilder automatisch erzeugen oder dynamisch Diagramme basierend auf aktuellen Umfragewerten ausgeben lassen.
- Alles, was Sie zur Arbeit mit PHP brauchen, steht kostenlos zur Verfügung. Im Buch werden Sie erfahren, wie Sie sich Ihre Entwicklungsumgebung mit wenigen Mausklicks einrichten.
- Webhosting-Angebote mit PHP-Unterstützung sind heute nicht mehr teuer.
- PHP ist weit verbreitet. Das bedeutet: Im Internet finden Sie auch bei spezielleren Fragen Hilfe und es gibt auch für ausgefallenere Anforderungen Lösungen.
- Viele bekannte Open Source-Anwendungen wie das Blogsystem Wordpress oder die Content Management-Systeme Joomla!, Drupal und TYPO3 basieren auf PHP.
- PHP ist für große Anwendungen wie Content Management-Systeme geeignet, aber auch für kleine: Wenn Sie die Daten aus dem Kontakt-Formular selbst verarbeiten, überprüfen und sich per Mail zusenden lassen wollen, ist PHP ebenfalls die richtige Wahl.

- Für große Projekte gibt es inzwischen mehrere Frameworks, die auf PHP aufsetzen.
- PHP macht Spaß!

Sie sehen, viele Gründe sprechen dafür, PHP zu lernen.

Welche Vorkenntnisse brauchen Sie, wenn Sie PHP lernen möchten? Mit PHP erstellen Sie dynamisch HTML-Seiten, die dann an den Browser ausgeliefert werden (Genaueres dazu in Kapitel 1). Deswegen sollten Sie über grundlegende HTML-Kenntnisse verfügen. Einen Crashkurs dazu gibt Ihnen Kapitel 3, aber falls Sie noch keine HTML-Seite erstellt haben, sollten Sie für die Einarbeitung in HTML zusätzliche Zeit einplanen und sich noch mit einem speziellen HTML-Buch eindecken. Prinzipiell sollten Sie außerdem Lust haben, sich in die Welt der Programmierung hineinzudenken.

Was erfahren Sie in diesem Buch?

In Kapitel 1 geht es erst einmal um die Grundlagen von PHP – Sie erfahren, was der Unterschied zwischen statischen HTML-Seiten und dynamisch per PHP erzeugten Seiten ist. Kapitel 2 zeigt Ihnen, wie Sie auf Ihrem Computer eine Entwicklungsumgebung installieren. Außerdem erfahren Sie am Beispiel, wie Sie PHP konfigurieren. Kapitel 3 gibt Ihnen im Schnelldurchlauf die wichtigsten HTML/CSS-Basics. In Kapitel 4 geht es um die Sprachelemente von PHP: Sie erfahren, wie Sie PHP in HTML-Dateien einbetten sowie welche Datentypen und Operatoren es gibt. Kapitel 5 führt weitere wichtige Sprachelemente ein – Sie lernen, Ihre Programme mit Bedingungen und Schleifen flexibel zu gestalten und Funktionen selbst zu erstellen. In Kapitel 6 sehen Sie wichtige fertige Funktionen, die Ihnen PHP zur Verfügung stellt: Mit diesen lassen sich Texte auf jede erdenkliche Art bearbeiten oder Arrays manipulieren und eigene Funktionen sind speziell für die Arbeit mit Datum und Uhrzeit gedacht.

Möchten Sie mit Ihren Benutzern kommunizieren, so ist ein wichtiger und klassischer Weg über Formulare. Kapitel 7 vermittelt Ihnen die wichtigsten Techniken zu Formularen und Sie erfahren auch, wie – und warum – Sie diese absichern müssen. Auch erfahren Sie am Beispiel, wie sich ein Bild-Upload per Formular realisieren lässt.

Cookies und Sessions sind eine weitere Webtechnologie: Mit Cookies und Sessions können Sie Zustände speichern. Das brauchen Sie beispielsweise, um Warenkörbe zu realisieren. Den Details zu Cookies und Sessions widmet sich Kapitel 8.

Durch die objektorientierte Programmierung lassen sich Programme besser warten und einzelne Komponenten leichter wieder verwenden. Kapitel 9 widmet sich detailliert der Objektorientierung und zeigt auch fortgeschrittene Möglichkeiten auf.

Wenn Sie mit umfangreichen Daten arbeiten, diese verändern möchten und auslesen, so empfiehlt sich der Einsatz einer Datenbank. Kapitel 10 liefert Ihnen die wichtigsten MySQL-Grundlagen. Sie erfahren, wie Sie mit phpMyAdmin arbeiten und lernen zudem, die wichtigsten MySQL-Befehle selbst zu schreiben. Das brauchen Sie dann in Kapitel 11, wenn es darum geht, wie Sie per PHP auf MySQL-Datenbanken zugreifen.

Nicht immer sind die Daten, die man bearbeiten möchte, in einer Datenbank gespeichert, manchmal liegen sie auch in Textdateien vor. Kapitel 12 zeigt Ihnen, wie Sie sowohl Inhalte aus Textdateien auslesen können als auch wie Sie per PHP in Textdateien schreiben können. Außerdem erfahren Sie, wie Sie einfach über die Schnittstelle simpleXML auf XML-Dateien zugreifen können, um beispielsweise Newsfeeds von anderen Seiten in Ihre Seite zu integrieren. Ein weiteres Thema sind die mit PHP 5.3 neu eingeführten Phar-Archive.

PHP kann mehr als Texte bearbeiten – Sie können mit PHP auch dynamisch Grafiken erzeugen oder vorhandene Bilder bearbeiten. Wie das geht, sehen Sie in Kapitel 13 anhand von zwei Beispielen: Sie erfahren, wie Sie kleine Vorschaubilder von größeren Bildern automatisch erstellen lassen und wie Sie Diagramme dynamisch realisieren.

Bisher wurden immer der (X)HTML- und der PHP-Code gemischt. Um diese zu trennen, gibt es so genannte Template-Systeme. Ein Beispiel für ein Template-System – Smarty – lernen Sie in Kapitel 14 kennen. Den Abschluss bildet der Anhang schließlich mit Informationen zu Möglichkeiten, PHP zu konfigurieren.

Den gesamten Code der Listings können Sie auf der Verlagswebsite zu diesem Buch unter *http://www.addison-wesley.de/9783827327239.html* herunterladen.

Jetzt aber wünsche ich Ihnen viel Spaß mit PHP!

1 Das Prinzip von dynamischen Webseiten

PHP ist eine serverseitige Skriptsprache. Was aber bedeutet das genau?

Serverseitig heißt erst einmal, dass der PHP-Code nicht auf dem Client, d. h. im Browser ausgeführt wird, sondern *auf dem Server*. Um das besser nachvollziehen zu können, muss man sich einmal vor Augen führen, wie die Kommunikation im Internet bei statischen HTML-Seiten ohne PHP-Code abläuft.

Diese besteht im Wesentlichen aus zwei Schritten:

1. Der Surfer gibt eine Adresse in die Adresszeile seines Browsers ein und drückt auf ENTER. Der Browser stellt eine Anfrage (REQUEST) an den Server nach der entsprechenden HTML-Datei.
2. Der Server liefert als Antwort (RESPONSE) diese Datei an den Browser, der sie dann darstellt.

In diesem klassischen Fall liegt die HTML-Seite, die der Browser anzeigt, genau in dieser Form auch auf dem Server. Der Server liefert die Datei, die in einem seiner Verzeichnisse liegt, nur aus, er verändert nichts daran.

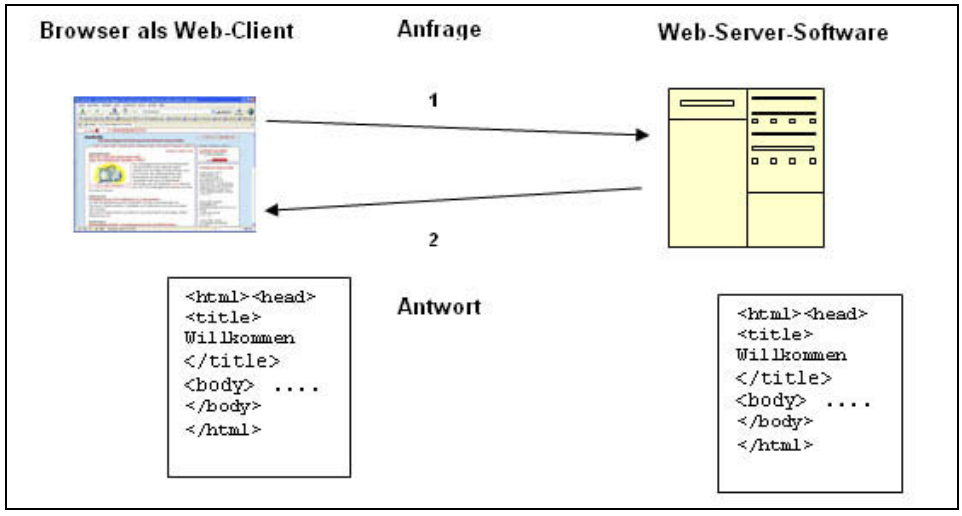


Abbildung 1.1: Kommunikation zwischen Browser und Webserver: Die angeforderte HTML-Seite wird an den Browser ausgeliefert, der diese darstellt.

Achtung

Übrigens hat das Wort *Server* zwei Bedeutungen. Zum einen meint man mit Server einen Rechner im Internet, d. h. die Hardware. Entscheidend aber dafür, dass die Kommunikation funktioniert, ist die *Webserver-Software*. Diese hat die Funktion, die Dateien auf Anfrage auszuliefern. Da die wichtige Komponente hier die Server-Software ist, können Sie – wie im nächsten Kapitel beschrieben –, sich auch einen Server auf Ihrem normalen Arbeitsrechner einrichten. Äußerst nützlich zum Testen von PHP-Skripten!

Sehen wir uns jetzt an, wie das Ganze funktioniert, wenn PHP mit im Spiel ist.

Dieses Mal sind es mehr Schritte:

1. Ein Surfer tippt eine Adresse ein. Der Browser leitet diesen `REQUEST` an den Webserver weiter.
2. Bei der angeforderten Seite handelt es sich um ein PHP-Skript. Der Webserver erkennt das an der Dateiendung *.php*. Daher liefert er die Seite nicht direkt an den Client aus, sondern übergibt sie einem Programm – bei PHP dem PHP-Parser.
3. Der PHP-Parser interpretiert die PHP-Befehle und erzeugt daraus eine neue HTML-Seite.
4. Diese HTML-Seite wird an den Browser zurückgesendet. Die im Browser angezeigte Seite heißt zwar noch *xy.php*, sie enthält aber keinen PHP-Code mehr.

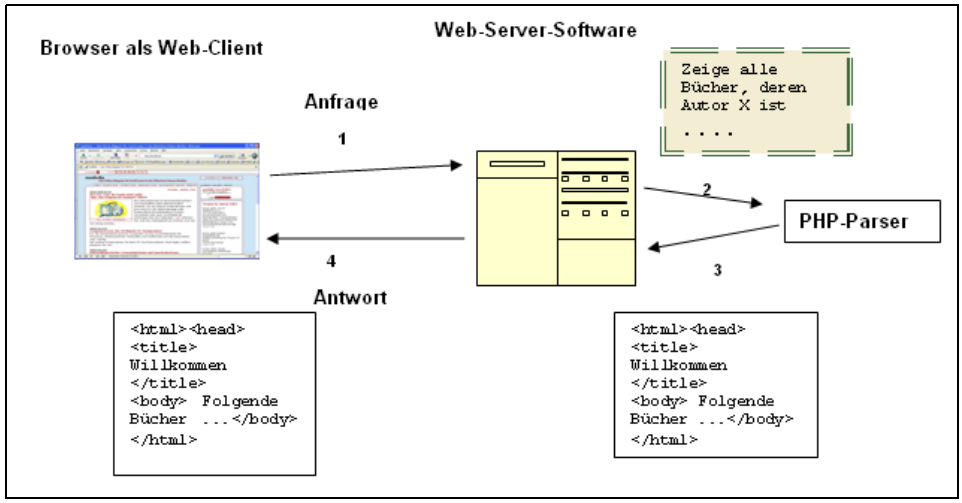


Abbildung 1.2: Die Kommunikation zwischen Client und Server bei dynamischen Seiten

Der entscheidende Unterschied zu den statischen Seiten ist, dass bei Anforderung einer PHP-Seite vom Browser in dieser PHP-Seite auf dem Server noch nicht die fertige HTML-Seite, sondern PHP-Befehle enthalten sind. Der Server reicht die Seite mit den PHP-Befehlen an den PHP-Parser weiter, der die Befehle ausführt und die neue HTML-Seite generiert. Und damit sind dynamische Seiten möglich, Seiten, die jedes Mal, wenn sie von einem Client aufgerufen werden, mit aktuellen Daten erstellt werden.

Da ein zusätzlicher Bearbeitungsvorgang auf dem *Server* stattfindet, spricht man davon, dass PHP eine *serverseitige Skriptsprache* ist. Für den Browser ändert sich hingegen nichts: Er erhält wieder eine einfache HTML-Datei. Das ist auch genau das, was der Browser versteht: Er kann nur HTML-Code darstellen, PHP kann er nicht interpretieren.

Hinweis

Da dynamische Webseiten bei Bedarf erstellt werden, ist es prinzipiell nicht möglich, zu sagen, wie viele Seiten im Internet stehen. Angenommen, Sie suchen ein Buch von Patricia Highsmith in Ihrem Online-Buchladen. Sie geben den Namen der Autorin in ein Formular ein und erhalten danach die Ergebnisse präsentiert. Falls noch niemand vor Ihnen in diesem Online-Buchladen nach Büchern von Patricia Highsmith gesucht hat, hat auch noch niemand genau die Ergebnisseite präsentiert bekommen, die Sie gerade sehen. Oder aber jemand hat vielleicht vor Ihnen schon genau diesen Suchbegriff eingegeben, aber seitdem sind neue Bücher erschienen – dann hat der Besucher vor Ihnen ebenfalls eine Seite mit anderen Ergebnissen gesehen.

Da PHP eine *serverseitige* Skriptsprache ist, steht es beispielsweise im Gegensatz zum *clientseitig*, das heißt im Browser ausgeführtem JavaScript. JavaScript wird z. B. eingesetzt, um Pop-up-Fenster zu öffnen oder um Formulareingaben zu prüfen. Da JavaScript im Browser ausgeführt wird, kann es vom Benutzer im Browser deaktiviert werden.

JavaScript kann wunderbar mit PHP kombiniert werden. Beispielsweise kann man eine Formularprüfung parallel mit JavaScript und PHP durchführen. Die Prüfung per JavaScript findet statt, bevor die Formulardaten den Rechner des Surfers verlassen, der Benutzer erhält ein schnelles Feedback. Da sich JavaScript jedoch deaktivieren lässt, findet sicherheitshalber eine zusätzliche Prüfung per PHP statt, die vom Benutzer nicht »ausgehebelt« werden kann.

Im nächsten Kapitel erfahren Sie, wie Sie sich Ihre Entwicklungsumgebung einrichten. Davor kurz aber noch die wichtigsten Eckpunkte zur Geschichte von PHP. Der Schöpfer von PHP ist Rasmus Lerdorf. Heute wird PHP von mehreren Entwicklern betreut. Die erste Version von PHP erschien 1995. Damals wurde die Abkürzung PHP noch aufgelöst als *Personal Home Page Tools*, inzwischen steht PHP wie erwähnt für *PHP Hypertext Preprocessor*. Die zweite Version von PHP erschien 1996, die dritte

1998. PHP 4 gibt es seit 2000 und im Juli 2004 ist PHP 5 erschienen. Erste Schnappschüsse von PHP 6 liegen bereits vor. Viele der ursprünglich für PHP 6 geplanten Features wurden in die Version 5.3 verlegt – die aktuelle Version, um die es hier im Buch geht.

2 Entwicklungsumgebung einrichten

Zur Arbeit mit PHP benötigen Sie eine entsprechende Entwicklungsumgebung. Dieses Kapitel zeigt Ihnen, wie Sie diese installieren. Außerdem erstellen Sie ein erstes PHP-Beispielskript und sehen, wie Sie die PHP-Konfiguration anpassen können.

2.1 Verschiedene Entwicklungsumgebungen

Um PHP-Skripte zu erstellen, brauchen Sie zwei Dinge:

- Die Webserver-Software – am häufigsten benutzt wird hier Apache
- PHP selbst

Wenn Sie dann – wie später im Buch beschrieben – auf eine Datenbank zurückgreifen möchten, brauchen Sie zusätzlich

- MySQL als Datenbankmanagementsystem.

Eine Möglichkeit ist, dass Sie Ihre Skripte bei einem Provider mit PHP-Unterstützung testen. Dann erstellen Sie Ihre Skripte lokal auf Ihrem Computer und laden sie zum Testen per FTP-Programm auf den Webserver beim Provider.

Praktischer ist es jedoch, wenn Sie sich selbst auf Ihrem lokalen Rechner eine vollständige Entwicklungsumgebung einrichten. Das hat mehrere Vorteile:

- Das Testen geht schneller vonstatten.
- Außerdem können Sie sich mit der Konfiguration von PHP vertraut machen und diese bei Bedarf auch anpassen – das ist eventuell beim Provider nur mit Einschränkung möglich.
- Zusätzlich können Sie Ihr Skript unter verschiedenen Bedingungen ausprobieren.

Die drei benötigten Komponenten – Webserver, PHP und MySQL – können Sie einzeln herunterladen und installieren. Es gibt jedoch praktische Komplettpakete, die alle benötigten Komponenten schon enthalten und die Installation wesentlich vereinfachen. Besonders erfolgreich ist XAMPP von den Apache Friends *http://www.apachefriends.org/de/*. XAMPP gibt es für Windows, Linux und Mac OS. Neben den unbedingt benötigten Komponenten beinhaltet XAMPP weitere nützliche Dinge wie

beispielsweise phpMyAdmin zur Administration von MySQL-Datenbanken (Genaueres zu phpMyAdmin in Kapitel 10). Deswegen wird hier die Installation von XAMPP gezeigt.

Hinweis

Um PHP mit dem Server zu verbinden, können Sie entweder eine direkte Modulschnittstelle benutzen oder PHP als CGI oder FastCGI-Prozessor benutzen. Die erste Variante wird bei XAMPP eingesetzt und ist prinzipiell aus Performanzgründen zu bevorzugen.

2.2 XAMPP-Installation unter Windows

XAMPP gibt es für Windows in verschiedenen Versionen, als Version mit Installer, als reine ZIP-Datei oder als selbst extrahierendes ZIP-Archiv. Hier soll beispielhaft die Installation als reine ZIP-Datei gezeigt werden.

Laden Sie sich die aktuelle ZIP-Datei unter *http://www.apachefriends.org/de/xampp-windows.html* herunter und entpacken Sie die Datei an eine beliebige Stelle.

Achtung

Nehmen Sie unter Vista nicht das Verzeichnis *c:\program files* (*c:\Programme*) zur Installation, da Sie dort standardmäßig keine Schreibrechte besitzen. Und das Gemeine dabei: Falls Sie versuchen, die XAMPP-Dateien dorthin zu extrahieren, erhalten Sie keine entsprechende Fehlermeldung: Nur die Zeit, die zum Kopieren der Dateien benötigt wird, ist hoch angegeben. Benutzen Sie ein anderes Verzeichnis wie *c:\xampp* oder *c:\unterordner\xampp*.

Es entsteht ein neuer Ordner mit Namen *xampp*.

Um die Pfade anzupassen, klicken Sie doppelt auf die Datei *setup_xampp*, die Sie im *xampp*-Ordner vorfinden. Damit werden die Pfade in den Konfigurationsdateien an Ihre Umgebung angepasst.

Als Nächstes sollten Sie die benötigten Programme starten. Dies können Sie komfortabel über die Datei *xampp-control.exe* erledigen, die Sie ebenfalls im Ordner *xampp* finden.

Klicken Sie bei APACHE und MYSQL auf START. Apache und MySQL werden gestartet (Abbildung 2.1). Über dieses Bedienfenster können Sie ebenso einzelne Programme wieder stoppen.

Tipp

Sie können Apache und MySQL auch als Dienst starten. Dann laufen diese Prozesse im Hintergrund. Hierfür müssen Sie das Häkchen bei SVC machen.

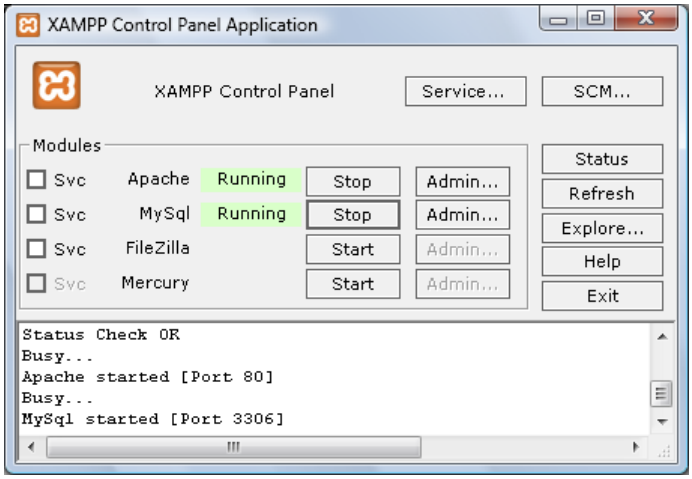


Abbildung 2.1: Das Kontrollzentrum von XAMPP unter Windows

Meist werden Sie Apache und MySQL benötigen, über das Control Panel können Sie ebenfalls die Server FILEZILLA und MERCURY starten.

FileZilla ist gleichzeitig der Name eines FTP-Clients und eines FTP-Servers. Über das XAMPP Control Panel können Sie den FTP-**Server** starten. Diesen brauchen Sie, wenn Sie auf Ihren eigenen Server Dateien per FTP laden möchten, beispielsweise weil Sie anderen Rechnern in einem Netzwerk die Möglichkeit bieten wollen, per FTP Daten zu übertragen. Wenn Sie XAMPP auf dem Rechner installieren, mit dem Sie auch die PHP-Dateien erstellen, benötigen Sie den FileZilla-Server nicht: Sie können Ihre PHP-Dateien direkt in das richtige Verzeichnis abspeichern.

Tipp

Um später Ihre Skripte zum Server des Providers hochzuladen, brauchen Sie hingegen einen FTP-Client – hier ist das Client-Programm von FileZilla empfehlenswert. Sie finden es unter *http://www.filezilla.de/*.

Mercury ist der Mailserver. Sie benötigen ihn, wenn Sie per PHP Mails versenden möchten (Kapitel 7).

Bei der Installation erhalten Sie mehrfach die bei Vista üblichen Warnungen (Benutzerkontenschutz), ob Sie wirklich die entsprechende Aktion ausführen möchten und müssen bestätigen, dass Sie es wirklich wollen.

Tipp

Normalerweise ist die Inbetriebnahme von XAMPP unproblematisch. Sollten Sie dennoch einmal Probleme haben, ist die FAQ zu XAMPP unter Windows eine gute Anlaufstelle: *http://www.apachefriends.org/de/faq-xampp-windows.html*.

2.3 XAMPP für Linux

Selbstverständlich können Sie bei allen gängigen Linux-Distributionen die benötigten Komponenten – Apache, PHP und MySQL – einzeln installieren. Aber auch hier bietet XAMPP eine Arbeitserleichterung und ist die richtige Wahl für alle, die sofort einsteigen und nicht erst konfigurieren möchten. Wenn Sie sich für XAMPP unter Linux entscheiden, finden Sie das Paket unter *http://www.apachefriends.org/de/xampp-linux.html*.

Wenn Sie es heruntergeladen haben, müssen Sie es noch entpacken. Öffnen Sie eine Konsole und werden Sie zu root über

```
su
```

Zum Entpacken dient folgender Befehl – ersetzen Sie die `x` in `xampp-linux-x.x.x.tar.gz` durch die Zahlen Ihrer Version:

```
tar xvfz xampp-linux-x.x.x.tar.gz -C /opt
```

Damit wird XAMPP unter `opt/lampp` installiert.

Hinweis

LAMPP war der ursprüngliche Name des Projekts und stand für Linux, Apache, MySQL, PHP, Perl. Ihm entsprach WAMPP auf der Windows-Seite. Um beide unter einem Begriff zusammenfassen zu können, heißt das Projekt jetzt XAMPP.

Zum Starten dient der Befehl:

```
/opt/lampp/lampp start
```

Die Meldungen in Abbildung 2.2 zeigen, dass Apache und MySQL gestartet wurden.

Abbildung 2.2: Es hat geklappt ...

Eben haben Sie gesehen, wie Sie XAMPP starten. Entsprechend können Sie XAMPP auch stoppen über:

```
/opt/lampp/lampp stop
```

Oder über folgende Zeile einen Neustart von XAMPP durchführen:

```
/opt/lampp/lampp restart
```

2.4 XAMPP für Mac OS X

Für Mac OS X sind die Schritte zur Installation ähnlich wie unter Linux. XAMPP für Mac OS X ist allerdings erst lauffähig ab Mac OS 10.4. Nach dem Herunterladen des Pakets (*http://www.apachefriends.org/de/xampp-macosx.html*) müssen Sie zuerst unter SYSTEMPROGRAMME die KONSOLE aufrufen und dann Folgendes eingeben:

```
sudo su
```

Nach der Eingabe des Passworts können Sie das Paket über folgenden Befehl entpacken – ersetzen Sie die `0` in `xampp-macosx-0.0.0.tar.gz` durch die Zahlen Ihrer Version:

```
tar xfvz xampp-macosx-0.0.0.tar.gz -C /
```

Zum Starten dient:

```
/Applications/xampp/xamppfiles/mampp start
```

Auf dem Bildschirm informieren nun die folgenden Zeilen über den Start des Systems:

```
Starte XAMPP für MacOS X 0.7.3...
XAMPP: Starte Apache mit SSL...
XAMPP: Starte MySQL...
XAMPP: Starte ProFTPD...
XAMPP gestartet.
```

Mit `/Applications/xampp/xamppfiles/mampp` können Sie XAMPP für Mac OS X stoppen:

```
/Applications/xampp/xamppfiles/mampp stop
```

Oder neu starten:

```
/Applications/xampp/xamppfiles/mampp restart
```

Tipp

Ganz ohne Befehle auf der Konsole einzugeben, geht es auch mit einem anderen Komplettpaket für Mac: *http://mamp.info/*. Das Praktische an dieser Lösung ist, dass keinerlei Systemdateien verändert werden und Sie auch zur »Deinstallation« nur den entsprechenden Ordner löschen müssen.

2.5 XAMPP testen

Sind die Programme gestartet, können Sie XAMPP austesten. Öffnen Sie dafür einen Browser und geben Sie in die Adresszeile `http://localhost` ein. Es erscheint der Startbildschirm von XAMPP, in dem Sie die Sprache auswählen können.

Abbildung 2.3: Dann hat es geklappt: So sieht der Startbildschirm von XAMPP aus.

Nach einem Klick auf die Sprache Ihrer Wahl kommen Sie zu einem Fenster, wo Sie sich genauer über die aktuelle XAMPP-Installation informieren können. Der Link STATUS im linken Navigationsbereich zeigt, welche Komponenten funktionieren.

Achtung

XAMPP ist gedacht für die Entwicklung von dynamischen Webseiten. Es ist nicht gedacht als Produktivsystem und dazu nicht geeignet. Dafür ist es viel zu unsicher konfiguriert: Eine erste Überprüfung der besonders sicherheitskritischen Einstellungen können Sie über den Link SICHERHEITSCHECK vornehmen. Für ein Produktivsystem sollten Sie besser gleich auf ein anderes System zurückgreifen.

2.6 Erste Beispieldatei

Ihre Dateien, die Sie über den Server ausliefern möchten, müssen Sie in einem besonderen Verzeichnis abspeichern. Bei XAMPP befindet sich dieses Verzeichnis innerhalb des *xampp*-Ordners und heißt *htdocs*.

In diesem Verzeichnis gibt es schon verschiedene Dateien, die von XAMPP benötigt werden. Damit sich Ihre Dateien nicht mit diesen in die Quere kommen, erstellen Sie am besten einen neuen Ordner. Nennen Sie ihn *php-beispiele*. In diesem werden Sie später alle Ihre Dateien abspeichern.

Tipp

In den folgenden Erläuterungen gehe ich immer davon aus, dass der Ordner *php-beispiele* heißt. Falls Sie ihn anders benennen, müssen Sie die Angaben entsprechend modifizieren.

Erstellen Sie nun eine erste Testdatei. Sie benötigen dafür einen einfachen Editor. Fürs Erste genügt der Editor, der bei Ihrem Betriebssystem dabei ist. Unter Windows finden Sie einen einfachen Editor unter START/ALLE PROGRAMME/ZUBEHÖR/EDITOR. Bei Linux können Sie zu Kate, Kwrite oder Ähnlichem greifen.

Schreiben Sie folgenden Code in die Datei:

Listing 2.1: Die erste PHP-Datei (phpinfo.php)

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

Tipp

Übrigens finden Sie alle Listings auf der Verlagswebsite zum Buch unter *http://www.addison-wesley.de/9783827327239.html* zum Download.

Speichern Sie diese Datei in Ihren eben erstellten Ordner *php-beispiele* im Unterordner *htdocs* des *xampp*-Ordners mit dem Namen *phpinfo.php*. Wichtig ist, dass Sie der Datei die Endung *.php* geben. Damit Ihnen der Editor nicht *.txt* als Endung dranhängt, wählen Sie sicherheitshalber unter Windows bei DATEITYP ALLE DATEIEN aus.

Nun zum Code-Beispiel: `<?php` dient dazu, den PHP-Code einzuleiten. Entsprechend beendet `?>` den PHP-Code wieder. Innerhalb von `<?php` und `?>` stehen die PHP-Befehle. Der eingesetzte Befehl `phpinfo()` liefert eine Fülle an nützlichen Informationen über die aktuelle PHP-Installation.

Rufen Sie jetzt die Datei im Browser auf, indem Sie in die Adresszeile Folgendes eingeben:

```
http://localhost/php-beispiele/phpinfo.php
```

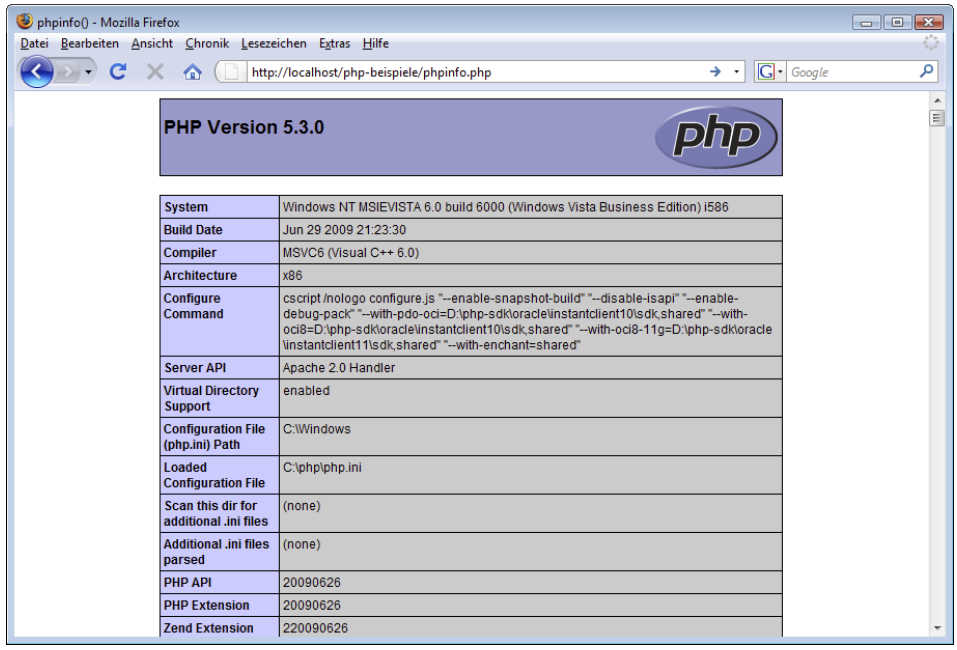
Das, was Sie über den Server erreichen, sind die Dateien des Ordners *htdocs*. Unterordner von *htdocs* geben Sie als Unterordner im Pfad an, im Beispiel den Unterordner *php-beispiele*. Danach folgt der Dateiname, hier *phpinfo.php*.

Ihre Ausgabe sollte wie in Abbildung 2.4 gezeigt aussehen.

Der Befehl `phpinfo()` ist sehr nützlich, um sich rasch einen Überblick darüber zu verschaffen, welche Version von PHP installiert ist und weitere Informationen über die Konfiguration, Pfade und installierten Module zu erhalten. Wenn Sie sich so wie hier beschrieben, PHP selbst installiert haben, könnten Sie diese Informationen theoretisch auch selbst auslesen, weil Sie ja Zugriff auf alle Konfigurationsdateien haben, aber das wäre relativ mühsam. Beim Hoster hingegen steht Ihnen diese Option oft nicht zur Verfügung, so dass der Befehl `phpinfo()` dort die einzige Möglichkeit ist, sich über PHP-Version und Installation zu informieren.

Achtung

Da diese Datei viele wichtige und grundlegende Dinge über PHP, Webserver etc. verrät, sollten Sie sie nicht unter dem Namen *phpinfo.php* auf einem echten Webserver im Internet belassen: Diese kann nämlich auch ein böswilliger Mensch aufrufen und erhält damit viele Informationen, die einen möglichen Angriff vereinfachen. Das heißt, Sie sollten die Datei bei einem echten Einsatz anders benennen und danach wieder löschen.



System	Windows NT MSIEVISTA 6.0 build 6000 (Windows Vista Business Edition) i586
Build Date	Jun 29 2009 21:23:30
Compiler	MSVC6 (Visual C++ 6.0)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--with-enchant=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626

Abbildung 2.4: Ausgabe von phpinfo() unter Windows – nützliche Informationen über die PHP-Installation und eindeutiges Indiz: Es hat alles geklappt.

Verdeutlichen wir uns noch einmal, was bei diesem Beispiel geschehen ist: Sie haben die PHP-Datei, die nur aus drei Zeilen besteht, im Webverzeichnis abgespeichert. Wenn Sie im Browser die Adresse `http://localhost/php-beispiele/phpinfo.php` eingeben, stellt der Browser eine Anfrage an den Webserver, die betreffende Datei auszuliefern. Der Webserver erkennt an der Endung *.php*, dass es eine PHP-Datei ist, die er nicht direkt ausliefern, sondern erst an die PHP-Engine weiterreichen muss. Diese bearbeitet den Befehl und erstellt eine HTML-Datei. Die HTML-Datei wird an den Browser ausgeliefert. Das ist auch genau das, was der Browser versteht: HTML kann er interpretieren und darstellen, PHP versteht er nicht.

Sehen Sie sich einmal den Quelltext der Datei im Browser an. Wählen Sie im Firefox ANSICHT/QUELLCODE, im Internet Explorer 7 SEITE/QUELLTEXT ANZEIGEN oder im Internet Explorer ANSICHT/QUELLE. Was sehen Sie? HTML und CSS-Code, das ist es auch schon.

Allerdings hat nicht jeder Befehl von PHP eine so große Wirkung – meist werden Sie den HTML-Code, den PHP ausgeben soll, selbst schreiben müssen. Deswegen brauchen Sie grundlegende Kenntnisse von HTML und CSS, um PHP richtig anweisen zu können, die HTML-Datei zu erstellen. Im nächsten Kapitel finden Sie einen kurzen Überblick über HTML/CSS, der Ihnen hierbei hilft. Dieser Überblick ist nicht vollständig, denn das Buch soll schließlich von PHP handeln. Aber Sie haben damit eine gute Basis, auf der Sie aufbauen können. Außerdem erfahren Sie, wo Sie bei HTML-

Fragen nachschauen können. Aber bevor wir dazu kommen, erst einmal zu möglichen Problemen, und danach dazu, wie Sie Ihre PHP-Installation anpassen können.

2.7 Mögliche Probleme beim Aufruf des ersten PHP-Dokuments

Falls der Aufruf Ihres ersten PHP-Dokuments nicht geklappt hat, finden Sie hier Hinweise, woran es liegen könnte.

Wenn Sie die Meldung OBJEKT NICHT GEFUNDEN! im Browser erhalten, haben Sie sich vielleicht verschrieben. Kontrollieren Sie dann, ob der Ordnername in der Adresszeile genauso heißt, wie Sie ihn auch im *htdocs* benannt haben. Und ob der Dateiname auch korrekt ist und mit *.php* beendet wurde. *htdocs* selbst darf in der Adresszeile nicht auftauchen. Denn mit `localhost` greifen Sie direkt auf den Inhalt von *htdocs* zu und mit `localhost/beispielordner/` auf den Inhalt, der innerhalb des Ordners *htdocs* im Unterordner *beispielordner* steht.

Abbildung 2.5: Hier stimmt die Adresse nicht: htdocs darf in der Adresszeile nicht angegeben werden.

Wenn Sie anstelle der gewünschten Ausgabe nichts sehen oder Kryptisches und im Quellcode PHP-Befehle entdecken, dann hat etwas Grundlegendes nicht geklappt. Wie im Beispiel könnte es sein, dass Sie versucht haben, die Datei direkt aufzurufen, ohne über den Server zu gehen.

Sie können PHP-Dateien nicht direkt aufrufen – also nicht wie bei HTML-Dateien durch Doppelklick oder durch DATEI/ÖFFNEN im Browser. In diesem Fall sehen Sie eine leere Seite und in der Quellcode-Ansicht den PHP-Code. Rufen Sie dann die Datei richtig auf, indem Sie am Anfang *localhost* und dann den richtigen Pfad angeben.

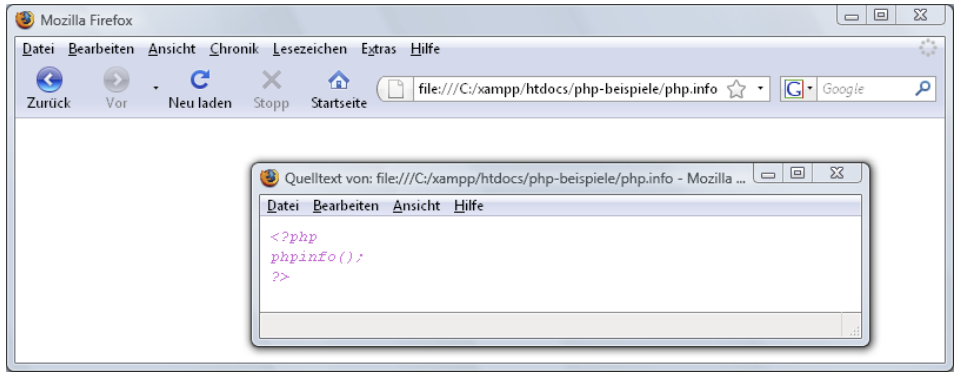


Abbildung 2.6: Eindeutiges Zeichen, dass etwas nicht geklappt hat.

Ein weiterer Fehler kann sein, dass die Endung nicht korrekt ist. Der Server ist so konfiguriert, dass er Dateien mit der Endung *.php* an die PHP-Engine weiterleitet. Steht zusätzlich noch *.txt* hinten dran oder haben Sie anstelle von *.php* die Endung *.html* verwendet, wird es ebenfalls nicht funktionieren.

Eine Meldung, dass die Verbindung fehlgeschlagen ist, ist hingegen ein Hinweis darauf, dass der Server nicht läuft. Starten Sie dann XAMPP.

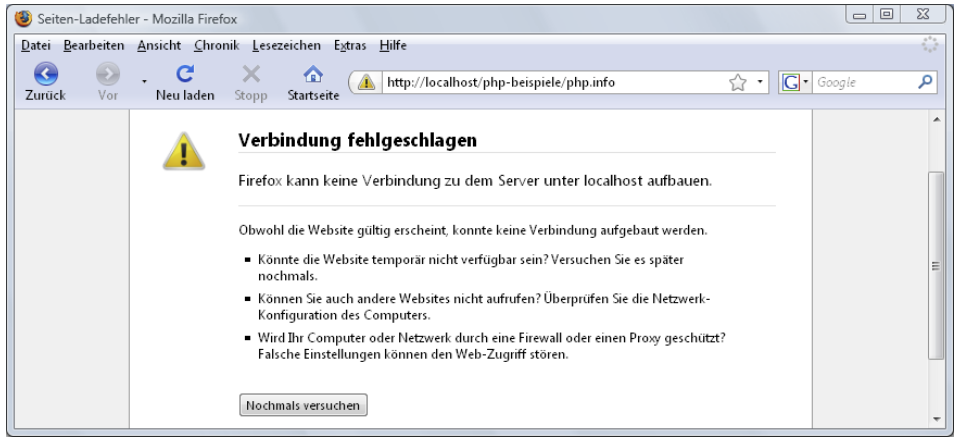


Abbildung 2.7: Wenn XAMPP nicht gestartet ist, erhält man diese Meldung.

2.8 PHP konfigurieren

Jetzt haben Sie eine Standardinstallation, die Sie an Ihre Bedürfnisse anpassen können. Die Konfiguration von PHP wird über eine Textdatei mit Namen *php.ini* gesteuert. Im Laufe des Buchs wird immer wieder von unterschiedlichen möglichen Konfi-

gurationen die Rede sein. In diesem Abschnitt sehen Sie an einem Beispiel, wie Sie die Konfigurationen ändern. Gezeigt wird das an der Einstellung für die Ausgabe der Fehlermeldungen.

Was die richtige Einstellung für die Ausgabe von Fehlermeldungen ist, hängt vom System ab. Auf einem Produktivsystem, also im echten Einsatz, sollten so wenige Fehlermeldungen wie möglich den Benutzer erreichen. Zum einen natürlich, weil ein Benutzer mit diesen Fehlermeldungen nichts anzufangen weiß. Zum anderen aber – und dieser Punkt ist weit wichtiger –, weil Fehlermeldungen aus Sicherheitsgründen gefährlich sind: Sie geben einem potenziellen Angreifer viele Informationen über Ihr System und mögliche Schwachstellen.

Das heißt: Keine PHP-Fehlermeldungen im Produktivbetrieb ausgeben lassen. Ganz anders sieht das bei der Entwicklung aus: Hier sollten Sie sich so viele Fehlermeldungen wie möglich anzeigen lassen. Denn Fehlermeldungen geben Ihnen wichtige Hinweise über mögliche Probleme mit Ihrem Code oder auf Stellen, an denen Sie sich einfach verschrieben haben.

Um die Ausgabe der Fehlermeldungen beeinflussen zu können, müssen Sie zuerst herausfinden, wo sich die Konfigurationsdatei von PHP befindet. Am besten konsultieren Sie hierfür die *phpinfo.php*-Datei. Suchen Sie (im Browser über Strg+F) nach *php.ini*.

Loaded Configuration File	C:\xampp\apache\bin\php.ini

Abbildung 2.8: Die Information über die benutzte php.ini-Datei findet sich über `phpinfo()`.

Damit wissen Sie, wo die benutzte *php.ini*-Datei gespeichert ist. Wechseln Sie dann in das angegebene Verzeichnis und wählen Sie die *php.ini*-Datei aus. Je nach Einstellung des Betriebssystems sehen Sie eventuell nur *php* als Dateiname, aber zusätzlich mit dem Hinweis, dass es sich um eine Konfigurationsdatei handelt.

Damit Sie bei Problemen jederzeit die ursprüngliche Datei wieder herstellen können, sollten Sie die *php.ini*-Datei zuerst kopieren. Öffnen Sie *php.ini* dann in einem Texteditor.

Die *php.ini*-Datei ist reichhaltig kommentiert. Kommentare sind immer mit einem Strichpunkt am Anfang versehen. In den Kommentaren finden Sie neben Erläuterungen der einzelnen Einstellungen oft auch Beispielkonfigurationen.

Wir wollen jetzt die Stelle finden, wo eingestellt ist, welche Fehler angezeigt werden sollen. Verwenden Sie die Suchfunktion Ihres Editors, um nach dem Wort »error« zu suchen. Sie kommen zum Abschnitt `; Error handling and logging ;`.

Hier erhalten Sie wichtige Hinweise, was Sie alles einstellen können und auch Beispielkonfigurationen angezeigt. Schließlich sehen Sie die aktuelle Einstellung – noch ein bisschen weiter unten und nicht mit einem Strichpunkt am Anfang der Zeile versehen. Diese sieht beispielsweise so aus:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

Das bedeutet, dass alle Fehler bis auf Hinweise angezeigt werden, oder sie sieht vielleicht so aus:

```
E_ALL & ~E_NOTICE & ~E_DEPRECATED
```

Was bedeutet, dass alle Fehler bis auf Hinweise und Deprecated-Meldungen angezeigt werden. Deprecated-Meldungen geben Hinweise, wenn Sie Sprachfeatures verwenden, die in PHP 5.3 schon veraltet sind und die es in PHP 6 nicht mehr geben wird.

Um auf dem Entwicklungssystem möglichst viele Fehler anzeigen zu lassen, ändern wir das in:

```
error_reporting  =  E_ALL | E_STRICT
```

Tipp

Das |-Zeichen finden Sie unter Windows übrigens auf Ihrer Tastatur bei den spitzen Klammern. Sie erreichen es durch zusätzliches Drücken von AltGr. Auf dem Mac geben Sie dieses Zeichen über ⌥ 7 ein.

Die geänderte Konfiguration sorgt dafür, dass auch die Hinweise ausgegeben werden. Das ist beispielsweise nützlich, wenn man sich bei Variablen verschreibt (mehr dazu in Kapitel 4). Über `E_STRICT` gibt PHP darüber hinaus noch Vorschläge für Änderungen des Programmcodes, die eine bestmögliche Interoperabilität Ihres Codes gewährleisten.

Speichern Sie die *php.ini*-Datei und schließen Sie sie. Die *php.ini*-Datei wird nicht sofort eingelesen, sondern erst wenn der Webserver neu gestartet wird. Deswegen müssen Sie den Apache neu starten. Am einfachsten erledigen Sie das in Windows über *xampp-control.exe*, indem Sie auf STOP und dann wieder auf START klicken. Für Mac und Linux benutzen Sie die in den entsprechenden Unterkapiteln erwähnten Befehle.

Nun sollten Sie testen, ob die Änderung wie gewünscht wirksam ist. Erstellen Sie dafür eine neue Datei mit folgendem Inhalt.

Listing 2.2: Drei Zeilen, um zu testen, ob die Anpassung der Konfiguration geklappt hat (test_fehlermeldungen.php).

```
<?php
  echo $hallo;
?>
```

Speichern Sie sie unter dem Namen *test_fehlermeldungen.php* in *php-beispiele* innerhalb von *htdocs*.

Abbildung 2.9: Ein Hinweis erscheint jetzt und zeigt, dass die Konfiguration geändert wurde.

Rufen Sie die Datei auf, indem Sie `http://localhost/php-beispiele/test_fehlermeldung.php` in die Adresszeile Ihres Browsers eingeben. Wenn Sie die oben angezeigte Fehlermeldung sehen, ist die veränderte Konfiguration wirksam. Sonst hätten wir keinen Hinweis erhalten, wenn wir eine Variable einsetzen, ohne sie definiert zu haben, jetzt hingegen sehen wir die Notice. Was es damit auf sich hat, erfahren Sie genauer in Kapitel 4.

2.9 Mehr PHP: Erweiterungen und PEAR

Wenn Sie XAMPP wie beschrieben verwenden, haben Sie eine gute Basis für die Entwicklung von PHP-Skripten. Die Standardeinstellungen sind bestens geeignet zum Testen.

Neben den grundlegenden Bestandteilen sind bei XAMPP auch die wichtigsten Erweiterungen aktiviert. Welche das sind, erfahren Sie ebenfalls über die Ausgabe von `phpinfo()`. Sollten Sie Erweiterungen brauchen, die nicht aktiviert sind, so sind zwei Schritte erforderlich: Sie brauchen erst einmal die gewünschte Bibliothek und müssen dann PHP noch mitteilen, dass diese Erweiterung berücksichtigt werden soll. Unter Windows geht der letzte Schritt recht einfach: In der *php.ini*-Datei muss der Kommentar vor der entsprechenden Zeile einfach entfernt werden. Unter Unix/Linux müssen Sie hierfür PHP neu kompilieren.

Tipp

Mit welchen Optionen die XAMPP-Version für Linux kompiliert wurde, sehen Sie in der Ausgabe von `phpinfo()` unter Linux bei CONFIGURE COMMAND.

Bei Ihrem Provider haben Sie eventuell keinen Zugriff auf diese Konfigurationsmöglichkeiten: Dann müssen Sie Ihren Provider bitten, die gewünschte Erweiterung zu aktivieren.

Neben den zu PHP gehörenden Erweiterungen gibt es weitere im PEAR. PEAR steht für PHP Extension and Application Repository (*http://pear.php.net/*) und ist eine Sammlung von PHP-Erweiterungen für die unterschiedlichsten und vielfältigsten Einsatzzwecke. Alle Projekte sind Open Source. Zu PEAR gehören zum einen diese Erweiterungen selbst, darüber hinaus ermöglicht PEAR aber auch die Verwaltung, d. h. Installation und Update der Pakete, per Kommandozeile. Das ist praktisch, da manche Pakete wiederum andere Pakete voraussetzen, d. h. voneinander abhängig sind. Diese Abhängigkeiten haben Sie über diese Verwaltungsmechanismen bestens im Griff.

Wenn Sie XAMPP installiert haben, erreichen Sie das Kommandozeilentool zur Verwaltung und Installation von PEAR-Paketen bei Windows unter `xampp\php\pear.bat`. Rufen Sie hierfür erst über START/ALLE PROGRAMME/ZUBEHÖR die EINGABEAUFFORDERUNG auf. Wechseln Sie in das Verzeichnis von `xampp\php`:

```
cd c:\pfadzurxamppinstallation\php
```

Wenn Sie dann `pear.bat` eingeben, erhalten Sie eine Liste von möglichen Kommandos.

Unter Linux geben Sie entsprechend `/opt/lampp/bin/pear` ein.

Am häufigsten wird der Befehl `install` *`Paketname`* benötigt, über den Sie ein einzelnes Paket installieren.

Bei einem Webhoster entfällt meistens diese Möglichkeit. Aber es gibt auch einen anderen Weg, PEAR-Pakete zu installieren. Hierfür rufen Sie zuerst die *http://pear.php.net/go-pear* auf, speichern den Inhalt als PHP-Skript ab und führen dieses Skript aus. Dieses Skript installiert Ihnen eine Verwaltungsoberfläche für PEAR-Pakete. Mehr Informationen zur Installation des PEAR-Managers finden Sie unter *http://pear.php.net/manual/de/installation.getting.php*.

Hinweis

Neben PEAR gibt es auch PECL: Die PHP Extension Community Library (PECL) war ursprünglich ein Teil von PEAR, ist aber inzwischen ausgegliedert. PECL enthält nur in der Programmiersprache C geschriebene Erweiterungen (*http://pecl.php.net/*).

Sie haben gesehen, was alles zur Entwicklungsumgebung von XAMPP dazugehört, wie sich Konfigurationseinstellungen ändern lassen und haben damit eine gute Vorbereitung für die weitere Arbeit mit PHP getroffen. Im nächsten Kapitel geht es jetzt aber wie angekündigt um die wichtige HTML/CSS-Basis.

3 HTML und CSS – Grundlagen

Webbrowser verstehen kein PHP. Webbrowser verstehen nur HTML und CSS. PHP setzen Sie ein, um HTML-Dokumente zu erstellen – mit dynamischen Inhalten. Deswegen sind auch für die Arbeit mit PHP grundlegende HTML-Kenntnisse notwendig. Diese vermittelt das Kapitel in Kurzform und verrät Ihnen, wo Sie weitere Informationen finden.

3.1 Grundstruktur

HTML steht für Hypertext Markup Language und ist eine Auszeichnungssprache für Webseiten. HTML wird vom World Wide Web-Konsortium – kurz W3C (*http://www.w3.org/*) – betreut. Das W3C ist ein unabhängiges Gremium, dem auch Vertreter von Firmen angehören.

HTML dient dazu, die Struktur einer Seite zu definieren und festzulegen, worum es sich bei den einzelnen Bestandteilen handelt, ob etwas ein Absatz, eine Aufzählungsliste oder eine Überschrift ist. Diese Bestandteile werden dann standardmäßig vom Browser auf eine bestimmte Art formatiert angezeigt. Um diese Default-Formatierungen zu ändern, kommt eine andere Technik ins Spiel: die Cascading Stylesheets (CSS), die in Abschnitt 3.7 besprochen werden.

Neben HTML gibt es auch XHTML, die erweiterbare Auszeichnungssprache (eXtensible Markup Language). XHTML unterscheidet sich in ein paar kleineren Punkten von HTML, zu denen kommen wir etwas später. Im Folgenden wird, wenn es um Dinge geht, die gleichermaßen für HTML und XHTML gelten, von (X)HTML gesprochen.

Jede (X)HTML-Seite besteht aus Tags und normalem Text. Tags werden in spitzen Klammern geschrieben, und man unterscheidet zwischen Start- und Endtags. `<html>` ist ein Starttag, das besagt, dass jetzt (X)HTML folgt. Dieses wird durch ein Endtag `</html>` geschlossen. Es gibt verschiedene Elemente, die auf bestimmte Art ineinander verschachtelt sein können, jedoch basieren alle (X)HTML-Dokumente auf demselben Grundgerüst:

Listing 3.1: Die Basis aller HTML-Dokumente: das (X)HTML-Grundgerüst (grundgeruest.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns="http://www.w3.org/1999/xhtml">
```

```
03 <head>
04   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
05   <title>Grundgerüst</title>
06 </head>
07 <body>
08   <p>Erste Webseite!</p>
09 </body>
10 </html>
```

Tipp

Zu besserer Orientierung sind im Folgenden längere Listings, d. h. im Allgemeinen Listings ab 10 Zeilen, mit Zeilennummern versehen. Diese gehören natürlich nicht zum Code und Sie dürfen sie nicht mit abschreiben.

Ganz am Anfang des Dokuments in Zeile 1 steht die Dokumenttypangabe. Diese etwas kryptisch anmutende Zeile verrät dem Browser, welche Version von (X)HTML in welcher Geschmacksrichtung eingesetzt wird. Im Listing ist es XHTML 1.0 in der Version Transitional. Darauf folgt das `html`-Start-Tag. Dieses hat bei XHTML noch das Attribut `xmlns` mit dem festgelegten Wert `http://www.w3.org/1999/xhtml`.

Alle Dokumente bestehen aus einem `head`- und einem `body`-Bereich. Der `head`-Bereich beinhaltet Informationen über das Dokument wie beispielsweise den Zeichensatz. Außerdem steht innerhalb des `title`-Elements der Titel des Dokuments. Dieser wird in der Titelzeile ganz oben im Browser angezeigt.

Innerhalb von `<body>` und `</body>` platzieren Sie die eigentlichen Inhalte, die im Browser dargestellt werden. Im Beispiel ist es ein `p`-Element, das zur Kennzeichnung eines Absatzes steht. Darin steht der Text, der angezeigt wird.

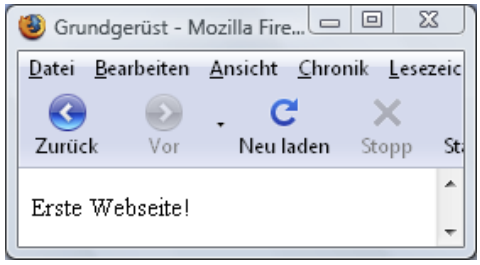


Abbildung 3.1: Ein erstes Dokument im Browser

Dieses Beispiel können Sie mit einem einfachen Texteditor erstellen. Geben Sie der Datei beim Speichern die Endung *.html*. Zum Betrachten mit dem Browser können Sie

sie direkt durch Doppelklick auf den Dateinamen öffnen – etwas, das ja bei PHP-Dokumenten nicht geht: Diese müssen Sie immer beim lokalen Test über den Server `localhost` aufrufen.

Tipp

Bei den Dateinamen haben Sie an sich freie Hand: Verwenden Sie aber keine Sonderzeichen wie ü, ä, ö oder ß und auch keine Leerzeichen. Da die Groß- und Kleinschreibung bei Betriebssystemen wie Linux relevant ist und viele Server im Internet auf Linux laufen, sollten Sie sich am besten angewöhnen, alles kleinzuschreiben.

3.2 HTML und XHTML und Strict und Transitional

Im Listing 3.1 wird XHTML 1.0 Transitional benutzt. Transitional ist die etwas laxere Variante, in der Sie vom W3C als veraltet/unerwünscht bezeichnete Elemente und Attribute noch einsetzen können. Es existiert daneben auch XHTML Strict, bei dem das nicht erlaubt ist.

Diese zwei Varianten Strict und Transitional gibt es ebenfalls bei HTML. Im Gegensatz zu XHTML, das auf XML (eXtensible Markup Language) basiert, basiert HTML auf SGML (Standard Generalized Markup Language). Die Regeln bei HTML sind etwas vager, es gibt häufiger Ausnahmen. Im Wesentlichen sind die Unterschiede zwischen HTML und XHTML syntaktischer Art: Beispielsweise müssen in XHTML im Gegensatz zu HTML alle Elemente geschlossen werden und zu jedem Starttag gibt es das passende Endtag.

Ein deutlicher Unterschied ist bei leeren Elementen zu sehen. Leere Elemente sind Elemente, die keine weiteren Elemente enthalten und auch keinen Textinhalt. Ein Beispiel haben Sie oben gesehen: das `meta`-Element zur Angabe des Zeichensatzes.

Dieses wird in XHTML so geschrieben:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
```

In HTML hingegen entfällt der abschließende Slash.

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

Im Folgenden wird noch auf weitere Unterschiede hingewiesen. Wenn etwas für HTML und XHTML gilt, wird die Schreibweise (X)HTML eingesetzt. Bei den Beispielen selbst wird aber immer XHTML eingesetzt.

Hinweis

Mehr Informationen über die Unterschiede zwischen HTML und XHTML lesen Sie unter *http://de.selfhtml.org/html/xhtml/unterschiede.htm*.

3.3 Inhalte strukturieren mit Überschirften, Absätzen und Listen

Das erste Beispiel war inhaltlich noch nicht sehr aufregend. Nun folgen weitere Elemente, die Sie zur Auszeichnung Ihrer Inhalte verwenden können. Das Element für Absätze `p` haben Sie bereits kennen gelernt. Möchten Sie nur einen Zeilenumbruch innerhalb eines Absatzes machen, so dient dafür `<br />`.

Tipp

`<br />` ist die XHTML-Schreibweise. In HTML würden Sie entsprechend `<br>` schreiben.

Zusätzlich gibt es vordefinierte Elemente für Überschriften. Es stehen sechs zur Verfügung: `h1` ist die Überschrift erster Ordnung, `h2` die Überschrift zweiter Ordnung etc. bis `h6`. Das folgende Listing führt die Überschriften `h1` bis `h3` vor. Es beinhaltet außerdem zwei Absätze und zeigt die Verwendung von `br` für Zeilenumbrüche.

Listing 3.2: Überschriften und Absätze dienen zur Strukturierung von Dokumenten (ueberschriften.html).

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06   <title>Strukturierungen</title>
07 </head>
08 <body>
09   <h1>Überschrift der ersten Ebene</h1>
10   <h2>Überschrift der zweiten Ebene</h2>
11   <h3>Überschrift der dritten Ebene</h3>
```

```
12   <p>Ein normaler Absatz<br />mit Zeilenumbruch</p>
13   <p>Ein normaler Absatz ohne Zeilenumbruch</p>
14 </body>
15 </html>
```

`<br />` wird verwendet, um einen Zeilenumbruch im Browser darzustellen. Leerzeichen und Zeilenumbrüche im (X)HTML-Quellcode selbst hingegen werden vom Browser ignoriert. Sie sollten diese aber einsetzen, um Ihren Code übersichtlich zu gestalten.

Wie Sie sehen, werden Überschriften automatisch fett dargestellt und – zumindest die Überschriften der Ebenen eins bis drei – auch größer als der Text in Absätzen. Das sind die Defaultvorgaben des Browsers. Per CSS können Sie das ganz nach Belieben anders gestalten.

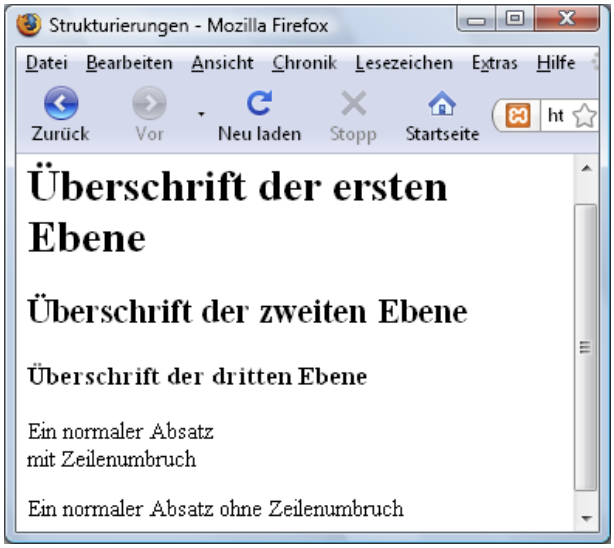


Abbildung 3.2: Überschriften und Absätze in einem Dokument

Suchen Sie eine Möglichkeit, einzelne Wörter oder Satzteile hervorzuheben, so sieht (X)HTML hier zwei Elemente vor: `strong` und `em` (*emphasized*).

```
<em>betont</em>
<strong>stärker betont</strong>
```

3.3.1 Aufzählungen

Weitere praktische Elemente sind Aufzählungen. Eine ungeordnete Liste erstellen Sie mithilfe des Elements `ul` (*unordered list*). Innerhalb von `<ul>` und `</ul>` stehen die einzelnen Aufzählungspunkte wiederum innerhalb von `<li>` und `</li>` (`li` = *list item*).

Für eine geordnete Liste, die automatisch vom Browser durchnummeriert wird, verwenden Sie anstelle von `ul` das Element `ol` (*ordered list*). Auch hier werden die einzelnen Punkte innerhalb von `<li>` und `</li>` notiert.

Listing 3.3: Listen im Einsatz (listen.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
  <title>Listen</title>
</head>
<body>
<ol>
  <li>Telefonate führen</li>
  <li><em>E-Mails</em> beantworten</li>
  <li>Stadtbibliothek: bestelltes <strong>Buch</strong> abholen</li>
  <li>Einkäufe</li>
</ol>
<ul>
  <li>Brot</li>
  <li>Weichkäse</li>
  <li>Buttermilch</li>
</ul>
</body>
</html>
```

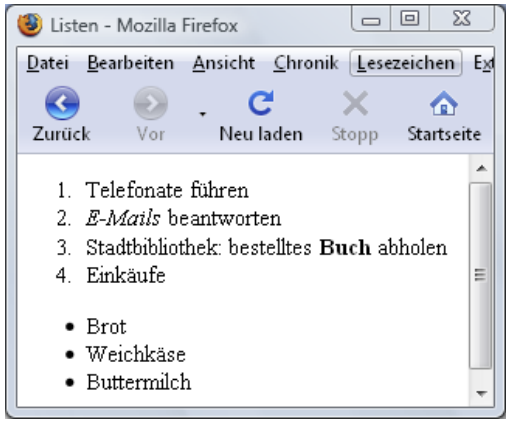


Abbildung 3.3: Eine nummerierte Aufzählung und eine Liste

Das Beispiel zeigt außerdem den Einsatz von `em` und `strong` für Betonungen. Normalerweise wird ein mit `em` markierter Teil kursiv, ein mit `strong` ausgezeichneter Textteil fett dargestellt. Aber auch das können Sie selbstverständlich per CSS ändern.

Bei komplexeren (X)HTML-Dokumenten ist es wichtig, Hinweise unterzubringen, die den (X)HTML-Code erläutern und einem helfen, später Modifikationen oder Ergänzungen vorzunehmen. Hierfür sind die Kommentare gedacht. (X)HTML-Kommentare beginnen Sie mit `<!--` und beenden Sie mit `-->`. Alles, was zwischen `<!--` und `-->` steht, wird vom Browser ignoriert.

```
<!-- Beginn Kopfbereich -->
```

3.4 Sonderzeichen und Zeichenkodierung

Wir haben bisher direkt ü's und Ä's geschrieben – geht das denn überhaupt gut, wenn jemand aus dem Ausland mit anderen Browser-Defaulteinstellungen auf unsere Seite geht? Ja, sofern Sie den Zeichensatz angeben, wie hier über eine `meta`-Angabe: Damit wählt der Browser automatisch die richtige Kodierung aus und auch die Sonderzeichen werden richtig dargestellt.

Hinweis

Die möglichen Zeichensätze werden später noch einmal bei der UTF-8-Unterstützung und den diesbezüglichen Besonderheiten von PHP besprochen (Kapitel 6).

Um ganz sicherzugehen, dass die Sonderzeichen unter allen Umständen korrekt angezeigt werden, können Sie aber auch die Entity-Schreibweise wählen. Entities beginnen immer mit einem `&`-Zeichen und enden mit einem Strichpunkt. Sie können den Wert des Zeichens dezimal oder hexadezimal angeben. Komfortabler und besser zu merken sind hingegen die benannten Entities. `Ü` steht beispielsweise für das große Ü. `Überschrift` gibt der Browser als »Überschrift« aus. Die wichtigsten Entities sehen Sie in Tabelle 3.1.

Zeichen	Benannte Entity	Zeichen	Benannte Entity
ä	`ä`	©	`©`
Ä	`Ä`	(geschütztes Leerzeichen)	` `
ö	`ö`	<	`<`
Ö	`Ö`	>	`>`
ß	`ß`	&	`&`
ü	`ü`	'	`'`

Tabelle 3.1: Die wichtigsten Entities

Zeichen	Benannte Entity	Zeichen	Benannte Entity
Ü	`Ü`	"	`"`
€	`€`	©	`©`

Tabelle 3.1: Die wichtigsten Entities (Forts.)

Wichtig sind die Entitities für die Zeichen < und `&`, da diese beiden eine Sonderbedeutung in (X)HTML haben: < leitet ein Tag ein, `&` steht am Beginn eines Entities. Wenn Sie diese Zeichen im normalen Text verwenden möchten, müssen Sie die entsprechenden Entities verwenden.

- `4 < 5` sollten Sie also schreiben als `4 < 5`.
- Für `Panther, Tiger & Co` schreiben Sie `Panther, Tiger & Co`.

3.5 Verknüpfungen – Links und Bilder

Das Internet wäre nicht das Internet ohne Verlinkungen und Bilder.

3.5.1 Links ...

Für Links gibt es das Element `a`, abgekürzt von englisch *anchor*. Wichtig ist beim `a`-Element das Attribut `href` (*hypertext reference*). Als Wert von `href` geben Sie den Dateinamen an, auf den Sie verlinken wollen.

Hinweis

Attribute dienen zur weiteren Kennzeichnung von (X)HTML-Elementen. In XHTML müssen die Werte von Attributen immer in Anführungszeichen stehen.

Der Text, den der Surfer sieht und auf den er klicken kann, um zum angegebenen Ziel zu kommen, steht innerhalb von `<a href="pfad">` und `</a>`:

```
<a href="ueberschriften.html">Dokument Überschriften aufrufen</a>
```

Befindet sich die Datei, auf die Sie verlinken wollen, im selben Verzeichnis wie Ihr Dokument, geben Sie direkt den Dateinamen an. Steht die Datei hingegen in einem Unterverzeichnis, so schreiben Sie dieses zuerst und den Dateinamen nach einem Slash (`/`):

```
<a href="dateien/dokument.html">Dokument im Unterverzeichnis </a>
```

Möchten Sie hingegen von einem Unterordner in den übergeordneten Ordner wechseln, notieren Sie zwei Punkte:

```
<a href="../ueberschriften.html">einen Ordner höher</a>
```

Sie können selbstverständlich auch auf eine andere Website im Internet verweisen. Hierfür müssen Sie die vollständige URL mit Protokoll (`http://`) angeben:

```
<a href="http://www.addison-wesley.de/">Addison-Wesley</a>
```

Standardmäßig werden Links im selben Fenster geöffnet. Soll die ursprüngliche Seite erhalten bleiben und die neue sich in einem neuen Fenster/Tab öffnen, so ergänzen Sie beim Link ein `target="_blank"`:

```
<a href="http://www.addison-wesley.de/" target="_blank">Addison-Wesley</a>
```

Übrigens hängt es dann von den Browser-Einstellungen ab, ob die neue Seite in einem neuen Fenster oder einem neuen Tab geöffnet wird.

Schließlich gibt es noch Links auf E-Mail-Adressen. Falls auf dem Rechner ein Mailprogramm installiert ist, öffnet sich dieses dann per Mausklick und die Adresse des Adressaten ist schon voreingetragen. Hierfür notieren Sie `mailto:` vor der E-Mail-Adresse:

```
<a href="mailto:ich@mir.de">E-Mail an ich@mir.de</a>
```

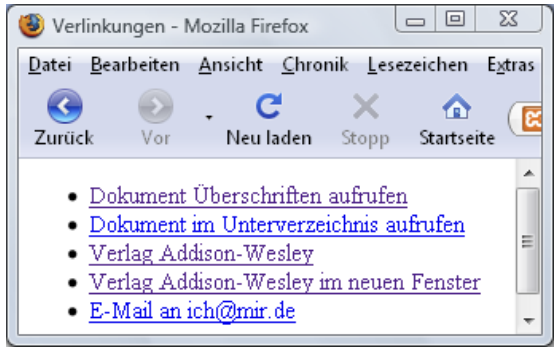


Abbildung 3.4: Eine Liste von Links

Listing 3.4 führt die einzelnen Linkarten noch einmal vor. Die Links werden – damit Sie sehen, wie Elemente verschachtelt werden können – innerhalb einer ungeordneten Liste dargestellt:

Listing 3.4: Eine Liste mit Links (verlinkungen.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
```

```
04 <head>
05 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06 <title>Verlinkungen</title>
07 </head>
08 <body>
09   <ul>
10     <li><a href="ueberschriften.html">Dokument Überschriften aufrufen</a></li>
11     <li><a href="dateien/weiteresdokument.html">Dokument im Unterverzeichnis
         aufrufen</a></li>
12     <li><a href="http://www.addison-wesley.de/">Verlag Addison-Wesley</a></li>
13     <li><a href="http://www.addison-wesley.de/" target="_blank">Verlag Addison-
         Wesley im neuen Fenster</a></li>
14     <li><a href="mailto:ich@mir.de">E-Mail an ich@mir.de</a></li>
15   </ul>
16 </body>
17 </html>
```

Links werden standardmäßig blau und unterstrichen dargestellt, die besuchten Links sind lila. All das können Sie per CSS abändern.

3.5.2 ... und Bilder

Das, was Sie gerade über die Pfade bei Links erfahren haben, gilt ebenfalls für Pfade zu Bildern. Wenn Sie in eine (X)HTML-Seite ein Bild einfügen möchten, so binden Sie das Bild nicht ein, sondern schreiben nur einen Verweis auf das Bild und überlassen das Einbinden dem Browser. Hierfür ist das `img`-Element vorgesehen, dem Sie beim Attribut `src` den Pfad zum Bild als Wert zuweisen:

```
<img src="blumen.jpg" />
```

Außerdem sollten Sie noch die Breite (`width`) und die Höhe (`height`) in Pixeln angeben und einen alternativen Text spezifizieren. Dieser alternative Text wird angezeigt, falls das Bild nicht geladen werden kann:

```
<img src="blumen.jpg" width="90" height="120" alt="Blumen" />
```

Im folgenden Listing werden zwei Bilder eingebunden.

Listing 3.5: Bilder einbinden per (X)HTML (bilder.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06 <title>Bilder</title>
07 </head>
08 <body>
```

```
09 <h1>Bilder einbinden</h1>
10 <img src="landschaft.jpg" width="90" height="120" alt="Landschaft" />
11 <img src="blumen.jpg" width="90" height="120" alt="Blumen" />
12 </body>
13 </html>
```

Abbildung 3.5: Ein Dokument mit zwei Bildern und einer Überschrift

3.6 Daten übersichtlich über Tabellen darstellen

Mit PHP kann man einfach auf Datenbanken zugreifen und die Inhalte nach bestimmten Kriterien gefiltert darstellen lassen. Die bevorzugte Darstellungsform für die Ausgabe von vielen Daten sind Tabellen. Sehen wir uns einmal an, wie man per (X)HTML eine Tabelle erstellt.

Eine Tabelle setzt sich aus mehreren verschachtelten (X)HTML-Elementen zusammen:

- `table` umfasst die gesamte Tabelle. Hier können Sie mit dem Attribut `border="1"` noch dafür sorgen, dass sichtbare Gitternetzlinien angezeigt werden. Das ist für den Entwurf praktisch, da dann Fehler besser zu erkennen sind. Sie können später bei Bedarf die Gitternetzlinien mit `border="0"` auch wieder ausschalten.
- `tr` (*table row*) umfasst jeweils eine Tabellenzeile.
- `td` steht für *table data*, umschließt den Inhalt der eigentlichen Tabellenzellen und wird innerhalb von `tr` notiert. Wenn Sie drei `td`-Elemente in einer `tr`-Zeile haben, so haben Sie eine dreispaltige Tabelle. Innerhalb von `td` steht der Inhalt der Zelle.
- Bei Spalten- oder Zeilenüberschriften sollten Sie für die einzelnen Zellen anstelle von `td` das Element `th` (*table header*) benutzen.

Das folgende Beispiel zeigt eine zweispaltige Tabelle mit drei Zeilen.

Listing 3.6: Tabellengrundgerüst (tabelle_grundgeruest.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06 <title>Tabelle - Grundgerüst</title>
07 </head>
08 <body>
09   <table border="1">
10     <tr>
11       <th>Abfahrt Hbh</th><th>Ankunft Katzenreuth</th>
12     </tr>
13     <tr>
14       <td>20:07</td><td>20:27</td>
15     </tr>
16     <tr>
17       <td>20:27</td><td>20:47</td>
18     </tr>
19   </table>
20 </body>
21 </html>
```

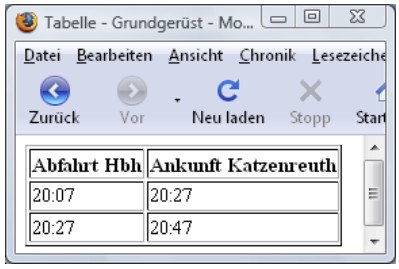


Abbildung 3.6: So sieht die Tabelle im Browser aus.

Über `rowspan` und `colspan` können Sie Tabellenzellen verbinden. Das Beispiel *zellen_verbinden.html* bei den Listings zu diesem Buch demonstriert das. Die Eigenschaften `cellpadding` und `cellspacing` sorgen für Abstand in der Tabelle, auch hierzu finden Sie in den Listings ein Beispiel (*cellpadding_cellspacing.html*).

3.7 Formatierung mit CSS

Zur Formatierung von Webseiten dient CSS. CSS steht für Cascading Stylesheets, also so viel wie kaskadierende Formatvorlagen. Es ist die Formatierungssprache für Webseiten und wird ebenfalls vom W3C betreut.

CSS-Regeln folgen immer einem klaren Schema. Zuerst steht der so genannte Selektor. Dieser wählt aus, für welche (X)HTML-Elemente die angegebene Formatierung gelten soll. Dahinter stehen in geschweiften Klammern die CSS-Anweisungen. Eine CSS-Anweisung besteht aus dem Namen der Eigenschaft, einem Doppelpunkt und einem vorgegebenen Wert. Abgeschlossen wird das mit einem Strichpunkt.

Wenn Sie als Selektor den Namen eines Elements schreiben – also beispielsweise `h1` oder `p` –, so gelten die angegebenen Formatierungen für alle entsprechenden Elemente. Die folgende Regel färbt alle `h1`-Überschriften rot ein und gibt ihnen eine gelbe Hintergrundfarbe.

```
h1 {
  background-color: yellow; color: red;
}
```

CSS-Regeln können Sie an verschiedenen Stellen notieren. Beim Entwurf ist es praktisch, die Angaben im Kopf des Dokuments zu notieren. Dafür schreiben Sie innerhalb des `head`-Bereichs `<style type="text/css"> </style>`.

Listing 3.7: Ein erstes CSS-Beispiel mit den CSS-Angaben im Dokumentkopf (css_anfang.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06 <title>Erste Formatierungen</title>
07 <style type="text/css">
08 h1 { background-color: yellow; color: red; }
09 p { background-color: gray; color: white; }
10 </style>
11 </head>
12 <body>
13 <h1>Eine Überschrift</h2>
14 <p>Und ein Absatz - dieses Mal bunt!</p>
15 <p>Noch ein Absatz</p>
16 </body>
17 </html>
```

3.7.1 Farbangaben

Oben haben Sie gesehen, dass für die Farbangabe ein englischer Farbname benutzt wurde. Offiziell erlaubt sind folgende sechzehn: `black` (schwarz), `green` (grün), `navy` (dunkelblau), `gray` (grau), `lime` (hellgrün), `blue` (blau), `maroon` (dunkelrot), `olive` (olivgrün), `purple` (violett), `red` (rot), `yellow` (gelb), `fuchsia` (magenta), `silver` (hellgrau), `aqua` (cyan), `teal` (blaugrün), `white` (weiß).

Abbildung 3.7: Überschrift und Absatz in verschiedenen Farben

Das ist gut zum Testen, aber für den richtigen Einsatz braucht man ein ausgefeilteres System. Im Web eingesetzt werden RGB-Farben (Rot-Grün-Blau), die üblicherweise über zweistellige Hexadezimalzahlen angegeben werden. Hexadezimale Ziffern reichen von 0 über 9 und A bis F. Der größtmögliche Wert ist damit FF, der kleinstmögliche 00.

Die Farbangaben bestehen aus sechs Stellen und werden direkt nach einem Gatterzeichen (#) notiert. Die beiden ersten Stellen geben den Rotwert, die beiden nächsten den Grün- und die beiden letzten den Blauanteil an. Damit lässt sich die Farbe Weiß als `#FFFFFF` schreiben, `#FF0000` beschreibt einen Rotton etc.

```
h1 { color: #FF0000; }
```

3.7.2 Mehr Freiheit durch Klassen

Bisher haben Sie gesehen, wie man Formatierungen für alle Elemente einer bestimmten Art vornimmt, beispielsweise für alle Absätze. Nicht immer möchte man aber alle gleichartigen Elemente gleich formatieren, sondern auch Ausnahmen definieren. Das lässt sich über Klassen realisieren.

Sehen wir uns das am Beispiel einer Zebratabelle an. Bei dieser ist jede zweite Zeile anders eingefärbt. Um das zu realisieren, vergibt man an einzelne Elemente, nämlich die, die anders formatiert werden sollen, *eine Klasse*. Das heißt: Man ergänzt, wo gewünscht, im (X)HTML-Teil das Attribut `class` mit einem frei wählbaren Namen:

```
<tr class="gerade">
```

Diese kann man dann bei der CSS-Definition ansprechen, indem man als Selektor den gerade vergebenen Namen mit einem Punkt davor notiert:

```
.gerade { background-color: #FFDFBF; }
```

Das folgende Beispiel zeigt die Realisierung einer Zebratabelle:

Listing 3.8: Bei dieser Tabelle erhält jede zweite Zeile eine Klasse und darüber eine andere Hintergrundfarbe (zebratabelle.html).

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06 <title>Zebratabelle</title>
07 <style type="text/css">
08 table { background-color: #FFC080; }
09 .gerade { background-color: #FFDFBF; }
10 </style>
11 </head>
12 <body>
13   <table border="1">
14     <tr>
15       <th>Abfahrt Hbh</th><th>Ankunft Katzenreuth</th>
16     </tr>
17     <tr class="gerade">
18       <td>20:07</td><td>20:27</td>
19     </tr>
20     <tr>
21       <td>20:27</td><td>20:47</td>
22     </tr>
23     <tr class="gerade">
24       <td>20:47</td><td>21:07</td>
25     </tr>
26     <tr>
27       <td>21:07</td><td>21:27</td>
28     </tr>
29   </table>
30 </body>
31 </html>
```

Im Beispiel werden zwei CSS-Regeln definiert: Zum einen erhält die ganze Tabelle eine Hintergrundfarbe (Zeile 8) und außerdem wird für die Klasse `.gerade` eine andere Hintergrundfarbe bestimmt (Zeile 9). Die Klasse wird dann bei ausgewählten `tr`-Elementen eingesetzt (Zeile 17 und Zeile 23).

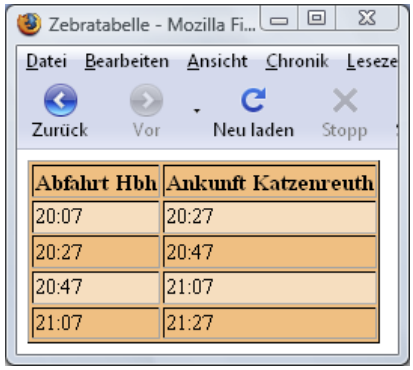


Abbildung 3.8: Zebratabelle – jede zweite Zeile ist in einer anderen Farbe eingefärbt.

3.7.3 Weitere häufig benötigte Formatierungen

Bisher wurden in den Beispielen für CSS-Formatierungen nur Farben eingesetzt. Aber es gibt natürlich mehr. Hier eine Auflistung häufig benötigter Formatierungen.

Schriftart: Die Schriftart geben Sie über `font-family` an. Als Eigenschaft notieren Sie eine Liste von Schriften. Ist die erste auf dem Computer des Surfers nicht vorhanden, nimmt der Browser die nächste usw. usf.

```
p { font-family: Verdana, sans-serif; }
```

Schriftgröße: Zur Angabe der Schriftgröße benutzen Sie `font-size`. Notieren Sie dabei die Einheit direkt an den Wert. Einsetzen können Sie `px` für Pixel oder `em`. Der genaue Wert eines em orientiert sich jeweils an der gewählten Schriftgröße und entspricht im Normalfall 16px.

```
h2 { font-size: 1.2em; }
```

Kursiv: Um etwas kursiv zu machen, notieren Sie `font-style: italic`.

Fett: Durch die Anweisung `font-weight: bold` wird etwas fett.

Ausrichtung: Über `text-align: center` zentrieren Sie Inhalte, `text-align: right` würde sie hingegen rechts anordnen.

Textausschmückungen: Mit `text-decoration` können Sie eine Unterstreichung (`text-decoration: underline`) bestimmen oder diese auch per `text-decoration: none` entfernen:

Ausmaße von Elementen: Mit `width` lässt sich die Breite von Elementen bestimmen, mögliche Einheiten sind `px`, `em` oder `%`: `width: 300px` macht ein Element 300px breit, `height` macht dasselbe für die Höhe. Mit `padding` schaffen Sie Abstand zwischen Elementen und ihrem Rand. `border` können Sie für einen Rahmen einsetzen.

Das folgende Listing zeigt die Formatierungen im Einsatz.

Listing 3.9: CSS-Formatierungen am Beispiel (formatierungen.html)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04 <head>
05   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
06   <title>Formatierungen</title>
07   <style type="text/css">
08   body { font-family: Verdana, sans-serif; }
09   .stil1 { font-style: italic; }
10   .stil2 { color: red; background-color: yellow;font-size:1.5em;}
11   .stil3 { font-weight: bold; text-align: center; }
12   .stil4 { border: 2px dotted blue; padding: 20px; }
13   .stil5 { width: 400px; height: 80px; background-color: orange;}
14   p { background-color: #DDDDDD; }
15   </style>
16 </head>
17 <body>
18 <p class="stil1">Hier steht kursiver Text (font-style: italic)</p>
19 <p class="stil2">Groß (font-size: 1.5em) und bunt (color: red; background-color:
yellow)</p>
20 <p class="stil3">Fett (font-weight: bold) und zentriert (text-align: center).</p>
21 <p class="stil4">Ein Absatz mit Rahmen und Innenabstand: border: 2px dotted blue;
     und padding: 200px</p>
22 <p class="stil5">Ein Absatz mit festgelegter Breite und Höhe (width: 400px;
     height: 80px; background-color: orange)
23 </body>
24 </html>
```

Im Beispiel wird für `body` die Schriftart Verdana oder eine andere serifenlose Schrift festgelegt (Zeile 8). Da `body` das umfassende Element ist, in dem alle anderen Elemente stehen, erben diese die Schriftart. Ab Zeile 10 beginnt die Definition von fünf Klassen, die immer unterschiedliche Formatierungen haben. Diese Klassen werden weiter unten innerhalb von `body` dann den einzelnen `p`-Elementen zugewiesen (Zeile 19–23). Außerdem wird allgemein für `p` eine graue Hintergrundfarbe definiert (Zeile 14).

Dieser kurze Überblick über (X)HTML/CSS hat aus Platzgründen natürlich nicht alle möglichen Elemente und Attribute behandelt. Suchen Sie weitere Informationen, so ist die Online-Dokumentation Selfhtml eine äußerst lohnenswerte Quelle. Sie finden sie unter *http://de.selfhtml.org/*. Sie können sich diese Dokumentation auch zum Offline-Lesen herunterladen.

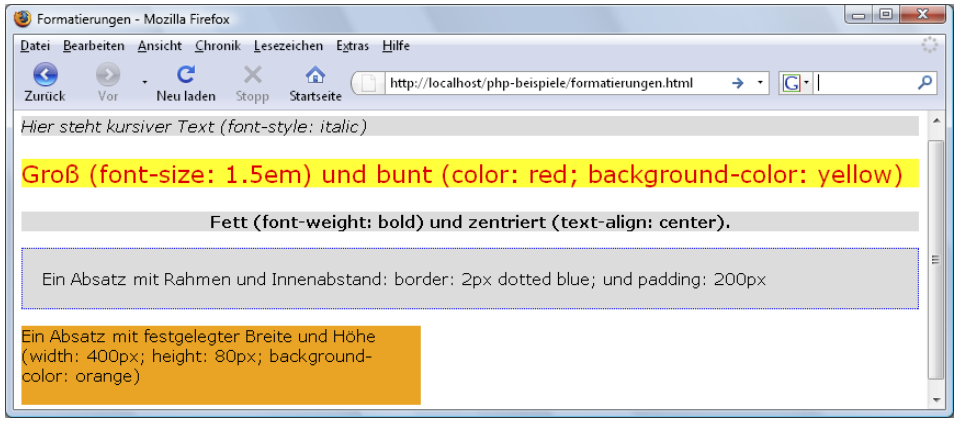


Abbildung 3.9: Das Formatierungsbeispiel

4 PHP-Basics

In diesem Kapitel erhalten Sie wichtige PHP-Grundkenntnisse. Sie erfahren, wie Sie PHP in (X)HTML-Dokumente einbinden und wie Sie mit Variablen Ihre Skripte flexibel halten. Außerdem geht es um unterschiedliche Datentypen und speziell um Arrays zum Speichern mehrerer Elemente. Zum Schluss sehen Sie, wie Sie mit PHP Dateien einbinden können – praktisch, um auf mehreren Seiten vorkommende Inhalte zentral zu speichern.

4.1 PHP in (X)HTML-Dokumente einbinden

PHP-Code können Sie direkt in (X)HTML-Dokumente einbinden. Damit der PHP-Parser die PHP-Befehle als solche erkennt, müssen diese innerhalb von `<?php` und `?>` notiert werden. Im folgenden Beispiel wird mit `echo` ein Text ausgegeben.

Listing 4.1: PHP-Code in ein XHTML-Dokument einbinden (php_einbinden.php)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>PHP in (X)HTML einbinden</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<?php
  echo "Ein erstes php-Dokument";
?>
</body>
</html>
```

Damit das Beispiel funktioniert, ist zweierlei notwendig: Die Datei muss im richtigen Verzeichnis abgespeichert sein und die Endung muss *php* sein. Falls es hierbei Probleme gibt, schauen Sie noch einmal nach in Kapitel 2.

Hinweis

.php ist die übliche und gängigste Endung für PHP-Dateien. Was als Endung bestimmt wird, lässt sich in der Konfiguration des Webservers festlegen: Sie könnten auch eine beliebige andere Zeichenkombination als Endung für PHP festlegen. Bei manchen Providern gibt es beispielsweise die Option, eine Endung wie *.php4* zu verwenden, wenn man möchte, dass die Skripte mit der veralteten Version 4 von PHP verarbeitet werden sollen.

Wenn Sie die Datei im Unterverzeichnis *php-beispiele* abgespeichert haben, rufen Sie sie über *http://localhost/php-beispiele/php_einbinden.php* in Ihrem Browser auf.

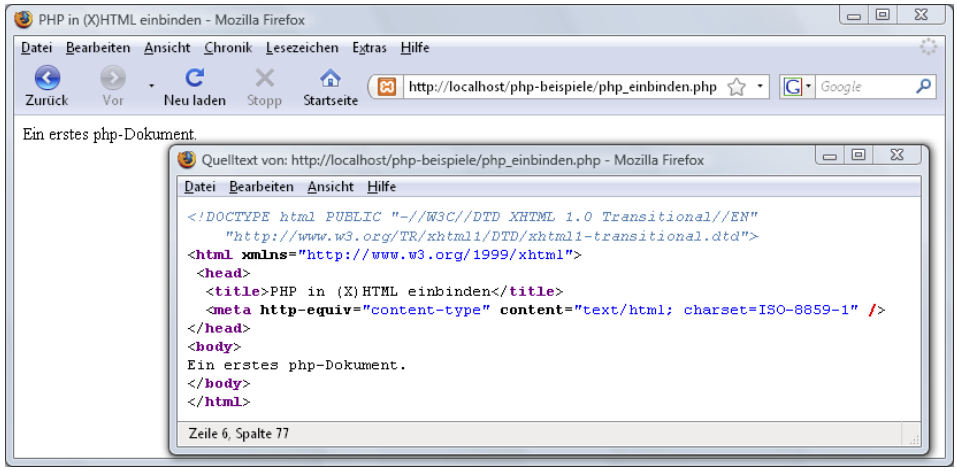


Abbildung 4.1: Dokument mit per PHP erzeugtem Text

Wechseln Sie dann einmal in den Quellcode: Hier sehen Sie keinen PHP-Code, sondern nur (X)HTML-Code. Wenn das so ist, hat alles geklappt.

Hinweis

Wir haben es hier mit zwei verschiedenen Quellcode-Dateien zu tun: Die Datei, die Sie in Ihrem Editor erstellt haben, enthält den PHP-Code, der mit (X)HTML-Code gemischt sein soll. Das, was Sie unter Quellcode in Ihrem Browser sehen, ist das, was der PHP-Interpreter auf dem Server erzeugt hat – ein reiner (X)HTML-Code ohne PHP-Befehle.

Jetzt genauer zum PHP-Code: Im Beispiel wird der PHP-Befehl `echo` eingesetzt, der zur Ausgabe dient. Handelt es sich um einen Text wie im Beispiel, müssen Sie diesen in Anführungszeichen schreiben: `echo "Unser erstes php-Dokument";`.

Hinweis

Für das, was hier allgemeinsprachlich mit Text bezeichnet wurde, gibt es die Fachbezeichnung »Zeichenkette« oder englisch »String«.

Außerdem sehen Sie am Ende einen Strichpunkt. Dieser dient in PHP dazu, Anweisungen abzuschließen.

4.1.1 Verschiedene Varianten der Einbindung

Im Beispiel wurde `<?php` und `?>` zum Einbinden des PHP-Codes benutzt. Das ist die gebräuchlichste und die empfohlene Variante, weil sie unabhängig von der Konfiguration immer funktioniert.

Es gibt daneben noch weitere Möglichkeiten:

- Mit dem `script`-Element:

```
<script language="php">
  echo "Eine andere Möglichkeit, PHP einzubinden";
</script>
```

- Eine Variante ohne das Wort `php` – das so genannte Short-open-Tag:

```
<? echo "noch mal hallo"; ?>
```

  Diese sehr kurze Option können Sie nur verwenden, wenn die Konfigurationseinstellung `short_open_tag` auf `On` steht. Ob das bei Ihrer Installation der Fall ist, können Sie in der Ausgabe von `phpinfo()` nachsehen.

short_open_tag	On	On

Abbildung 4.2: Wenn die entsprechende Zeile in der Ausgabe von phpinfo() so aussieht, würden die Short-Open-Tags ebenfalls funktionieren.

  Allerdings kommt es bei dieser Schreibweise zu Problemen, wenn Sie XHTML-Dokumente mit der XML-Deklaration am Anfang schreiben `<?xml version="1.0" ?>`.

- Eine weitere Möglichkeit der Einbindung sind die so genannten ASP-Tags:

```
<% echo "auch das ist möglich"; %>
```

  Ob ASP-Tags möglich sind, ist ebenfalls von der Konfiguration abhängig. Die entsprechende Einstellung heißt `asp_tags`.

Achtung

Wenn Sie sichergehen möchten, dass Ihre PHP-Skripten überall laufen, sollten Sie *nur* die klassische Form `<?php` und `?>` benutzen.

4.1.2 PHP-Befehle überall

Die PHP-Befehle können Sie an beliebigen Stellen in Ihrem (X)HTML einfügen – immer da, wo Sie sie brauchen. Also dort, wo Sie beispielsweise – wie später gezeigt – einen Wert aus der Datenbank ausgeben oder das Ergebnis einer Berechnung anzeigen lassen wollen:

Im folgenden Beispiel wird PHP-Code an mehreren Stellen eingefügt.

Listing 4.2: PHP-Code kann an sich überall stehen (php_code_ueberall.php).

```
01 <?php date_default_timezone_set("Europe/Berlin");?>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
03     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
04 <html xmlns="http://www.w3.org/1999/xhtml">
05  <head>
06   <title><?php echo date("j.n.Y"); ?></title>
07   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
08 <style type="text/css">
09   body { background-color: <?php echo "yellow"; ?>; }
10 </style>
11 </head>
12 <body>
13 <?php
14   echo "Schönen Tag auch!";
15 ?>
16 </body>
17 </html>
```

In Zeile 1 – sogar vor der XHTML-Dokumenttypangabe – steht ein erster Aufruf von PHP. Im Beispiel wird damit die Zeitzone gesetzt. In Zeile 6 folgt der nächste Aufruf von PHP: Hier wird im Seitentitel das Datum ausgegeben.

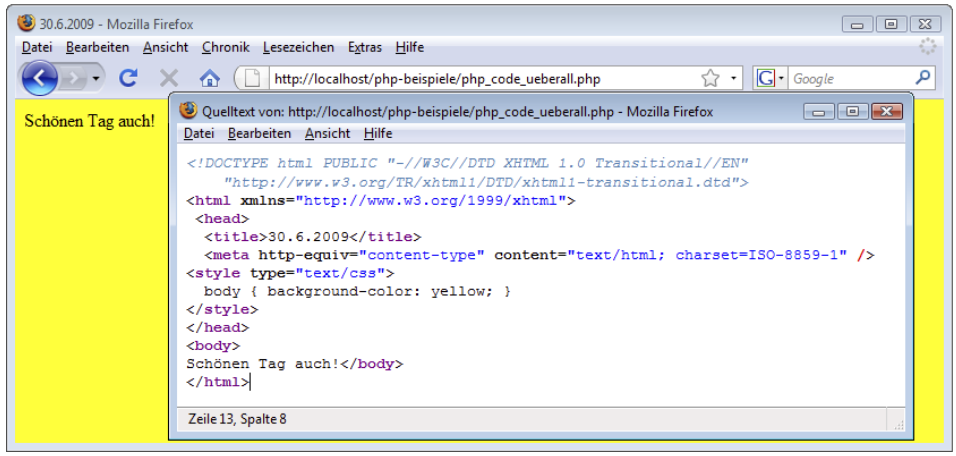


Abbildung 4.3: Das Ergebnis ist wieder korrektes (X)HTML.

Hinweis

Mehr zur Funktion, um die Zeitzone zu setzen, sowie zur `date()`-Funktion zur Datumsausgabe in Kapitel 6.

In Zeile 9 wird noch einmal PHP aufgerufen: Dieses Mal innerhalb der CSS-Angaben, bei der Zuweisung einer Hintergrundfarbe für das `body`-Element. Der letzte Aufruf von PHP ist dann in Zeile 13, wo eine Begrüßung ausgegeben wird. Das alles ist problemlos möglich. Wichtig ist nur, dass das Ergebnis wieder korrektes (X)HTML ist.

Umgekehrt können Sie natürlich auch (X)HTML-Code direkt innerhalb vom Text schreiben, der per `echo` ausgegeben wird:

```
<?php
  echo "<p>Schönen Tag auch!<br />Bis später</p>";
?>
```

In diesem Fall könnten Sie `<p>` und `</p>` auch außerhalb des PHP-Codes notieren – das macht keinen Unterschied:

```
<p>
<?php
  echo "Schönen Tag auch!<br />Bis später";
?>
</p>
```

Am Quellcode des (X)HTML-Dokuments, das ausgeliefert wird, können Sie prinzipiell nicht feststellen, welche Teile über PHP-Befehle erzeugt werden und welche direkt im (X)HTML-Code standen.

Tipp

Im Beispiel wurde immer `echo` eingesetzt. Stattdessen können Sie auch `print` benutzen:

```
<?php
  print "Schönen Tag auch!<br />Bis später";
?>
```

Ob Sie `echo` oder `print` wählen, ist im Wesentlichen Geschmackssache. Genauere Informationen zu den kleinen Unterschieden, die im Normalfall nicht wichtig sind, finden Sie unter *http://www.faqts.com/knowledge_base/view.phtml/aid/1/fid/40*.

Leerzeichen und neue Zeilen sind für PHP nicht relevant. Sie sind jedoch ganz essenziell für die Lesbarkeit des Skripts. Wie man diese geschickt einsetzt, erfahren Sie etwas später – in Kapitel 5 –, wenn Sie weitere PHP-Sprachelemente kennen gelernt haben.

Innerhalb von Anführungszeichen sind die Leerzeichen hingegen schon relevant, sie werden eins zu eins so in den ausgegebenen Quellcode übernommen. Hier sind die meisten aber nicht sichtbar, da Browser Leerzeilen im (X)HTML-Code ignorieren und mehrere Leerzeichen zu einem zusammenfassen.

4.2 Kommentare

Mit Kommentaren können Sie Erklärungen zu Ihrem Skript in den Quellcode schreiben, die vom PHP-Interpreter ignoriert werden. Vielleicht ist Ihnen heute bei einem Skript noch klar, warum Sie was an welche Stelle geschrieben haben, aber sehen Sie sich mal ein von Ihnen selbst geschriebenes Skript nach ein paar Monaten noch einmal an: Sie werden sich an wenig erinnern und froh sein, wenn Sie Hinweise finden, was die einzelnen Schritte bedeuten und warum sie durchgeführt wurden. Außerdem sind Kommentare ganz essenziell, wenn mehrere Personen an einem Skript arbeiten.

Kommentare können einzeilig sein:

```
//dies ist ein Kommentar
#dies ist auch ein Kommentar
```

Einzeilige Kommentare können auch als Anschluss an einen PHP-Befehl stehen:

```
echo "Hallo"; //gibt Hallo aus
```

Mehrzeilige Kommentare stehen zwischen /* und */:

```
/* dies ist ein
mehrzeiliger
Kommentar */
```

Kommentare können auch verwendet werden, um gerade nicht benötigte Codezeilen auszukommentieren. Im nächsten Beispiel wird die zweite Ausgabe auskommentiert:

```
<?php
  echo "<p>Schönen Tag auch!<br />Bis später</p>";
  /* echo "Der derzeitige Gesamtbetrag ist 42,50<br />"; */
  echo "Weitere interessante Produkte finden Sie unter ... ";
?>
```

Das kann man bei der Fehlersuche einsetzen, um festzustellen, ob die Fehlermeldung durch eine bestimmte Zeile/einen bestimmten Codebereich hervorgerufen wurde.

Achtung

Mehrzeilige Kommentare dürfen nicht verschachtelt werden. Das Folgende würde nicht funktionieren:

```
/* Das ist ein Kommentar
/* und hier fängt ein neuer Kommentar an */
Und erst hier wird der Kommentar beendet */
```

Das Ende des zweiten, im ersten verschachtelten Kommentars würde auch den ersten Kommentar beenden.

Prinzipiell verwendet man /* und /* für längere Kommentare zu Beginn eines Skripts oder eines Skriptbereichs, für die kleinen Schritte dazwischen hingegen //. In den Skripten in diesem Buch sehen Sie hingegen wesentlich häufiger die /* */-Kommentare, das liegt daran, dass die Zeilen hier kürzer sind als sonst.

4.3 Variablen definieren und ausgeben

Sie haben bisher gesehen, wie Sie über PHP Texte ausgeben lassen können. Viele zusätzliche Möglichkeiten ergeben sich durch ein ganz wichtiges weiteres PHP-Sprachelement: die Variablen. Variablen sind Platzhalter für unterschiedliche Daten – z. B. Text oder Zahlen – und nichts anderes als ein symbolischer Bezeichner für einen Speicherbereich, in dem ein Wert abgelegt wird. Variablen sind beispielsweise notwendig, um Eingaben der Benutzer weiterzuverarbeiten: Sie wissen ja noch nicht, was die Benutzer eingeben, möchten aber trotzdem darauf zugreifen, um die Inhalte beispielsweise auszugeben.

Variablennamen beginnen in PHP immer mit einem Dollarzeichen: `$meineVariable`.

Die Namen von Variablen vergeben Sie selbst. Dabei müssen Sie folgende Regeln beachten:

- Groß- und Kleinschreibung wird unterschieden. So sind `$meineVariable` und `$MeineVariable` unterschiedliche Variablen.
- Nach dem Dollarzeichen darf nicht direkt eine Zahl folgen: `$7kaese` wäre also kein korrekter Variablenname.
- Leerzeichen, Punkte, Ausrufezeichen oder Bindestriche sind in Variablennamen nicht erlaubt. Statt des Leerzeichens nehmen Sie am besten einen Unterstrich, z. B. `$brutto_preis`.

Um einer Variable einen Wert zuzuweisen, verwenden Sie den Zuweisungsoperator =:

```
$name = "Lola";
$alter = 2;
```

Achtung

= kennen Sie sicher auch aus der Mathematik. In der Mathematik bedeutet es »ist gleich«, hier in PHP hingegen »erhält den Wert«.

Variablen können Sie natürlich nicht nur einen festen Wert, sondern auch das Ergebnis einer Berechnung zuweisen.

```
$erg = 17 + 4;
```

4.3.1 Notice bei nicht initialisierten Variablen

Wenn Sie eine Variable einsetzen, der Sie keinen Wert zugewiesen haben, erhalten Sie eine entsprechende Notice. Allerdings nur, wenn Sie die Einstellung `error_reporting` wie in *Kapitel 2* beschrieben, geändert haben. Ein Beispiel:

Listing 4.3: Nicht initialisierte Variable (nichtinitialisiert.php)

```
$zahl = 5;
$erg = $Zahl + 10;
```

Hier wird `$zahl` der Wert `5` zugewiesen, dann aber in der Berechnung `$Zahl` (mit Großbuchstaben) eingesetzt. Da die Groß-/Kleinschreibung von Variablen relevant ist, sind `$zahl` und `$Zahl` für PHP verschiedene Variablen und `$Zahl` ist nicht initialisiert, das heißt, Sie haben ihr keinen expliziten Wert zugewiesen.

Abbildung 4.4: Fehlermeldung bei nicht initialisierter Variable

Sie erhalten dann die in Abbildung 4.4 gezeigte Meldung – das Skript würde ansonsten aber trotzdem funktionieren und der Variablen `$Zahl` der Defaultwert `0` zugewiesen. Die Fehlermeldung ist aber hier sehr hilfreich, da sie Ihnen einen Hinweis auf Ihren Tippfehler gibt.

4.3.2 Inhalt von Variablen ausgeben

Den Inhalt von Variablen können Sie per `echo` ausgeben:

```
echo $name;
```

Häufig möchte man Textinhalt mit dem Inhalt von Variablen kombinieren. Also nicht nur ein Wort ausgeben lassen, sondern einen ganzen Satz. Das geht denkbar einfach: Sie können direkt Text und Variablen bei der Ausgabe kombinieren:

```
echo "$name ist $alter Jahre alt.";
```

Gibt aus »Lola ist 2 Jahre alt.«

Dieser Vorgang, dass innerhalb einer Zeichenkette Variablennamen erkannt und durch ihren Wert ersetzt werden, heißt Variableninterpolation und wird nur durchgeführt, wenn Sie den Text in doppelten Anführungszeichen schreiben. Verwenden Sie stattdessen einfache Anführungszeichen, sehen Sie `$name` anstelle von `Lola` in der Ausgabe und `$alter` anstelle von `2`:

```
echo '$name ist $alter Jahre alt.';
```

Hinweis

Häufig müssen Sie nur schnell in PHP wechseln, um einen Wert ausgeben zu lassen:

```
<?php echo $wert; ?>
```

Genau für diesen Fall gibt es eine verkürzte Schreibweise. Sie schreiben direkt nach `<?` ein =-Zeichen und dann das, was Sie ausgeben lassen möchten:

```
<?=$wert?>
```

Kurz und praktisch – allerdings funktioniert diese Schreibweise nur, wenn die Konfigurationseinstellung `short_open_tag` auf `on` steht. Das heißt, wenn Sie das einsetzen, sind Sie abhängig von der Konfiguration, und das ist wiederum unpraktisch (siehe auch Abschnitt 4.1.1).

4.3.3 Sonderzeichen in Anführungszeichen

Möchten Sie z. B. innerhalb von doppelten Anführungszeichen wirklich ein Dollarzeichen ausgeben lassen, müssen Sie es »maskieren«: So stellen Sie sicher, dass PHP das Dollarzeichen als normales Dollarzeichen und nicht als Einleitung für eine Variable nimmt:

```
echo "Das Buch kostet 14 \$";
```

Genauso müssen Sie auch einen Backslash vor ein doppeltes Anführungszeichen setzen, wenn Sie es innerhalb von doppelten Anführungszeichen einsetzen wollen. Das werden Sie häufig brauchen bei Attributwerten in (X)HTML, die selbst in Anführungszeichen geschrieben werden:

```
echo "<img src=\"wiesen.jpg\" width=\"137\" height=\"103\" alt=\"Landschaft\" />";
```

Das ergibt dann als XHTML-Code:

```
<img src="wiesen.jpg" width="137" height="103" alt="Landschaft" />
```

Sieht man sich dann die Datei im Browser an, wird – sofern das Bild im Ordner vorhanden ist, – die Landschaft angezeigt.

Anstatt die doppelten Anführungszeichen über `\"` zu maskieren, können Sie auch einfache Anführungszeichen für die Attributwerte in (X)HTML verwenden.

```
echo "<img src='wiesen.jpg' width='137' height='103' alt='Landschaft' />";
```

Dies ließe sich auch umgekehrt schreiben – indem Sie außen die einfachen und innerhalb dieser die doppelten Anführungszeichen einsetzen.

```
echo '<img src="wiesen.jpg" width="137" height="103" alt="Landschaft" />';
```

Listing 4.4 fasst diese unterschiedlichen Verwendungen noch einmal zusammen.

Tipp

Bei diesem Beispiel wurde das umfassende XHTML-Grundgerüst nicht mehr mitabgedruckt. Das wird im Folgenden immer so gehandhabt, wenn der PHP-Teil ganz normal innerhalb von `<body>` und `</body>` steht.

Listing 4.4: Variablen ausgeben mit einfachen und doppelten Anführungszeichen (variablen_ausgeben.php)

```
$name = "Lola";
$alter = 2;
$erg = 17 + 4;
echo "<h3>Mit doppelten Anführungszeichen: </h3>";
echo "$name ist $alter Jahre alt.";
echo "<h3>Mit einfachen Anführungszeichen: </h3>";
echo '$name ist $alter Jahre alt.<br />';
echo "<h3>Und noch ein paar Bilder: </h3>";
echo "<img src=\"wiesen.jpg\" width=\"137\" height=\"103\" alt=\"Landschaft\"
   />";
echo "<img src='wiesen.jpg' width='137' height='103' alt='Landschaft' />";
echo '<img src="wiesen.jpg" width="137" height="103" alt="Landschaft" />';
```

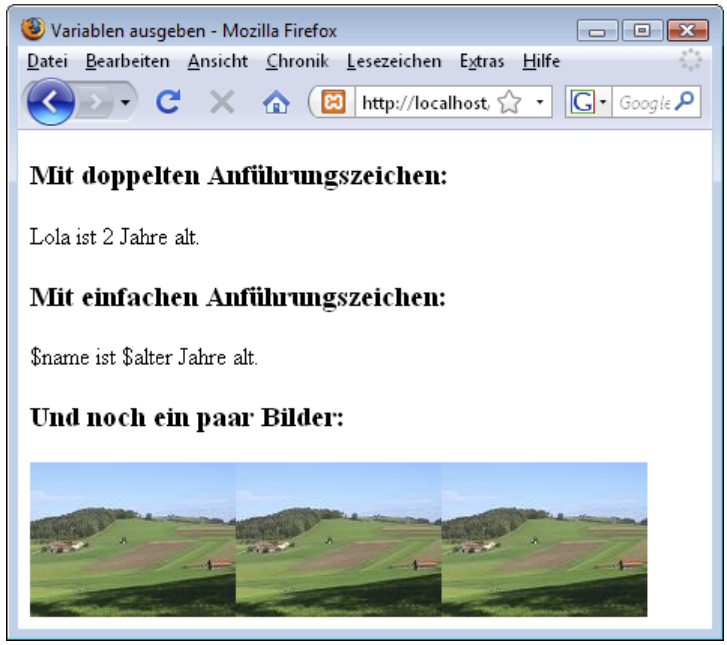


Abbildung 4.5: Unterschiedliche Verwendung von einfachen und doppelten Anführungszeichen

Sie haben gesehen, wie Sie den Backslash innerhalb von doppelten Anführungszeichen einsetzen können, um Sonderzeichen wie das `$`-Zeichen oder doppelte Anführungszeichen selbst auszugeben. Daneben gibt es weitere Kombinationen von Backslash und Zeichen, die innerhalb von doppelten Anführungszeichen eine besondere Bedeutung haben.

\n und \t für einen übersichtlichen (X)HTML-Quellcode

Ihren (X)HTML-Quellcode strukturieren Sie i. d. R. durch Zeilenumbrüche und Einrückungen. Um dies auch für den (X)HTML-Code zu machen, den der PHP-Interpreter aus den PHP-Befehlen erzeugt, verwenden Sie `\n` und `\t`. `\n` erzeugt einen Zeilenumbruch, `\t` einen Tabulator:

Listing 4.5: Tabulator und Newline im Einsatz (escapesequenzen.php)

```
echo "Unser erstes \nphp-Dokument. \n";
echo "\tUnser erstes \tphp-Dokument. \n";
```

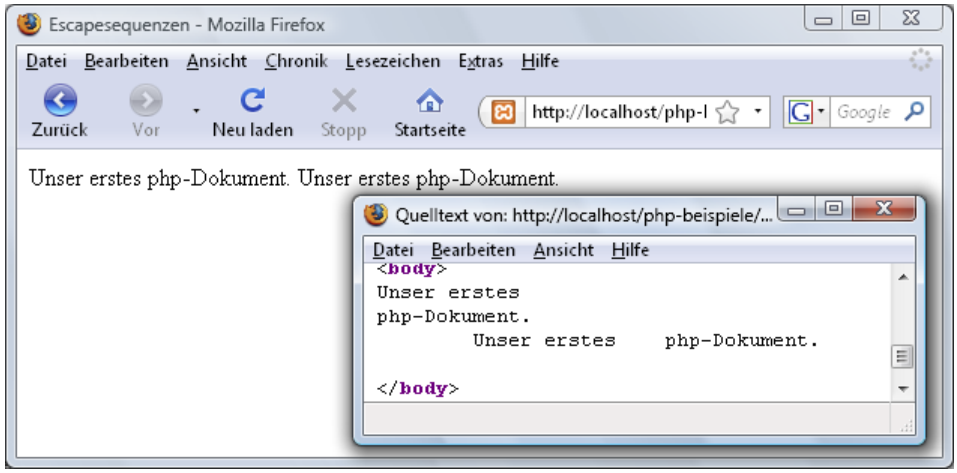


Abbildung 4.6: Auswirkungen zeigen \t und \n nur im (X)HTML-Quellcode, nicht in der Ausgabe des Browsers.

In Abbildung 4.6 sehen Sie deutlich, dass `\t` und `\n` aus dem PHP-Code keine Auswirkung im Browser haben, sondern nur im (X)HTML-Quellcode. Sinnvoll ist ihr Einsatz beispielsweise, wenn man mit PHP eine Tabelle ausgeben lässt. Hier kann man den Quellcode für eine bessere Lesbarkeit per `\t` und `\n` einrücken – das ist hilfreich, um mögliche Verschachtelungsfehler beim Einsatz von `<tr>` oder `<td>` zu finden.

Ein weiterer sinnvoller Einsatz von `\n` ist für die Erzeugung von Zeilenumbrüchen in Textdateien (siehe Kapitel 12) oder bei Textmails (Kapitel 7).

Wenn Sie in Ihrem Text selbst ein `\` benutzen, so müssen Sie dieses ebenfalls maskieren durch einen weiteren Backslash:

```
$windowspfad = "C:\\xampp";
```

Alle möglichen Escapesequenzen

Die Kombination von Backslash plus Zeichen wird Escapesequenz genannt. Alle möglichen Escapesequenzen führt Tabelle 4.1 vor: Innerhalb von einfachen Anführungszeichen gibt es nur zwei Escapesequenzen: `\'` für ein einfaches Anführungszeichen innerhalb von Anführungszeichen und `\\` für den Backslash innerhalb von einfachen Anführungszeichen selbst.

Kombination	Bedeutung
`"\\"`	\
`"\n"`	Neue Zeile
`"\t"`	Tabulator
`"\$"`	Dollarzeichen
`"\""`	"
`"\r"`	Wagenrücklauf
`"\v"`	Vertikaler Tabulator
`"\f"`	Seitenvorschub
`"\100"`	Das Zeichen, das der angegebenen Oktalzahl in der Codetabelle des Zeichensatzes entspricht – hier @.
`"\X40"`	Das Zeichen, das der angegebenen Hexadezimalzahl in der Codetabelle des Zeichensatzes entspricht – hier @.
`'\\'`	\
`'\''`	'

Tabelle 4.1: Escapesequenzen in einfachen und doppelten Anführungszeichen

4.3.4 Variablennamen über {} kennzeichnen

Noch eine Besonderheit bei der Variableninterpolation: Sie haben ja gesehen, dass Sie den Wert von Variablen direkt in doppelten Anführungszeichen ausgeben lassen können. Was aber, wenn man direkt an den Wert etwas dranhängen möchte, beispielsweise ein Genetiv-s?

```
$vorname= "Amina";
```

Und Sie möchten ausgeben lassen »Aminas Jacke«. Wenn Sie das so versuchen:

```
echo "$vornames Jacke";
```

versteht der PHP-Interpreter `$vornames` als Variablenname. Da Sie keine Variable mit diesem Namen definiert haben, wird nichts ausgegeben. Wenn wie in Kapitel 2 beschrieben die Anzeige der Fehlermeldungen so eingestellt ist, dass auch Hinweise (Notice) angezeigt werden, erhalten Sie eine Meldung, dass Sie eine nicht definierte Variable verwenden.

Aber es besteht natürlich eine Möglichkeit, etwas direkt an die Variable anzuhängen. Sie müssen PHP dabei nur mitteilen, wie weit der Variablenname geht und wo der zusätzliche Text ist. Dazu brauchen Sie geschweifte Klammern:

```
echo "{$vorname}s Jacke";
```

4.3.5 Komfortable Ausgabe über HereDoc und NowDoc

HereDoc und NowDoc sind eine weitere Möglichkeit zur Ausgabe von Text. Wenn Sie mehr (X)HTML-Tags und Variablen mischen wollen, ist das manchmal mühsam: Sie müssen immer darauf achten, die Anführungszeichen zu maskieren oder die jeweils anderen zu verwenden etc. Eine Vereinfachung kann hier die HereDoc-Syntax bringen.

Um etwas über HereDoc ausgeben zu lassen, schreiben Sie hinter `echo` <<< und einen Bezeichner, im Beispiel ist es `DOC`. Danach geben Sie Ihren (X)HTML-Code ganz »normal« an – Sie können beispielsweise Anführungszeichen ganz unmaskiert verwenden. Sie beenden die HereDoc-Syntax mit dem Bezeichner, mit dem Sie das Ganze begonnen haben, und einem Strichpunkt.

```
echo <<<DOC

DOC;
```

Achtung

Wichtig ist, dass der abschließende Bezeichner bei der HereDoc-Syntax ganz am Anfang der Zeile steht. Es darf kein Leerzeichen und auch kein anderes Zeichen davor stehen.

Das folgende Listing zeigt die HereDoc-Syntax zur Ausgabe einer Tabelle.

Listing 4.6: Ausgabe über die HereDoc-Syntax (heredoc.php)

```
01 $vorname = "Amina";
02 $alter   = 3;
03 echo <<<DOC
04 <table border="1" cellpadding="5" cellspacing="0">
05   <tr>
06     <td>Name</td>
07     <td>Alter</td>
08   <tr>
09   <tr>
10     <td>$vorname</td>
```

```
11     <td>$alter</td>
12   <tr>
13 </table>
14 DOC;
```

Sie müssen den Text nicht direkt ausgeben lassen, sondern können ihn auch in einer Variable speichern und später bei Bedarf ausgeben.

Listing 4.7: Dieses Mal wird der Text erst einmal in einer Variable gespeichert (heredoc_2.php).

```
01 $vorname = "Amina";
02 $alter = 3;
03 $ausgabe = <<<DOC
04 <table border="1" cellpadding="5" cellspacing="0">
05   <tr>
06     <td>Name</td>
07     <td>Alter</td>
08    <tr>
09   <tr>
10     <td>$vorname</td>
11     <td>$alter</td>
12   <tr>
13 </table>
14 DOC;
15 echo $ausgabe;
```

Der Text innerhalb der HereDoc-Syntax wird vom PHP-Interpreter so behandelt, als stünde er in doppelten Anführungszeichen – und Variablen werden interpoliert. Im Beispiel erscheinen nach der Verarbeitung anstelle von `$vorname` der zugewiesene Wert *Amina*. Genau darin unterscheidet sich eine andere mögliche Konstruktion mit Namen NowDoc von HereDoc. Bei NowDoc wird der Inhalt so behandelt, als stünde er in einfachen Anführungszeichen und der Wert der Variablen wird nicht ausgegeben. NowDoc steht erst ab PHP 5.3 zur Verfügung.

NowDoc definieren Sie wie HereDoc mit dem Unterschied, dass Sie den Bezeichner in einfachen Anführungszeichen schreiben.

Listing 4.8: NowDoc (nowdoc.php)

```
01 $vorname = "Amina";
02 $alter = 3;
03 echo <<<'DOC'
04 <table border="1" cellpadding="5" cellspacing="0">
05   <tr>
06     <td>Name</td>
07     <td>Alter</td>
08    <tr>
```

```
09   <tr>
10     <td>$vorname</td>
11     <td>$alter</td>
12   <tr>
13 </table>
14 DOC;
```

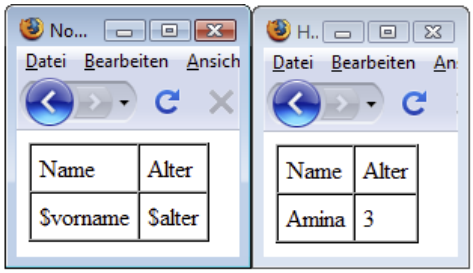


Abbildung 4.7: Links NowDoc-, rechts HereDoc-Syntax

4.3.6 Qual der Wahl: einfache oder doppelte Anführungszeichen?

In diesem Unterkapitel ging es um die Definition von Variablen und um die Ausgabe von Texten und Variablen. Dabei macht es ja einen Unterschied, ob Sie die doppelten oder die einfachen Anführungszeichen wählen. Was soll man jetzt im konkreten Fall jeweils nehmen – einfache oder doppelte Anführungszeichen? Der Unterschied ist ja bekanntlich, dass bei doppelten der Wert von Variablen ausgegeben wird, bei einfachen nicht.

Wenn man diesen Unterschied streng berücksichtigt, sollte man natürlich nur dann doppelte Anführungszeichen verwenden, wenn sie benötigt werden.

- Typischer Fall für doppelte Anführungszeichen: `echo "Hallo $name";`
- Eher ein Fall für einfache Anführungszeichen: `echo "Guten Morgen";`

Andererseits ist es – nach meiner Erfahrung aus Kursen – für PHP-Einsteiger relativ umständlich und mitunter verwirrend, wenn sie bei allen Ausgaben immer zuerst überlegen müssen, welche Anführungszeichen denn nun angebracht sind.

Deswegen werden hier im Buch konsequent doppelte Anführungszeichen eingesetzt, und wenn innerhalb dieser weitere benötigt werden – z. B. bei Attributwerten bei (X)HTML-Tags –, einfache benutzt. Diese Regel lässt sich durchgehend anwenden und funktioniert immer.

4.3.7 Voll flexibel: variable Variablen

In PHP können Sie Variablennamen selbst in Variablen speichern und darüber auf die Variablen zugreifen. Hierfür benutzen Sie zwei Dollarzeichen:

Listing 4.9: Variable Variable (variable_variablen.php)

```
$varname = "beispiel";
$$varname = "php";
echo $beispiel;
```

Hier wird eine Variable mit Namen `$varname` definiert und mit dem String `beispiel` als Inhalt. Dann erhält `$$varname` den Inhalt `php`. Die Ausgabe von `echo $beispiel` ist dann »php«.

4.4 Konstanten definieren

Der Inhalt von Variablen ist, wie der Name sagt, variabel, er kann sich im Laufe des Skripts ändern. Wenn Sie hingegen mit feststehenden Werten in Ihrem Skript arbeiten, sollten Sie Konstanten einsetzen. Konstanten definieren Sie nicht über Zuweisung wie Variablen, sondern über die Funktion `define()`. In runden Klammern geben Sie zuerst den Namen der Konstanten an, nach einem Komma den Wert.

Hinweis

`define()` ist eine von PHP vorgegebene Funktion. PHP stellt Ihnen viele solcher vordefinierten Funktionen zur Verfügung, die Sie direkt einsetzen können. Hinter dem Funktionsnamen stehen runde Klammern, in denen Sie PHP die Parameter für die Funktion übergeben. Mit Parametern bestimmen Sie, mit was die Funktion operieren soll. Mehrere Parameter werden dabei durch Komma voneinander getrennt. Wie viele Parameter Sie angeben können und wie viele Sie müssen, ist von Funktion zu Funktion unterschiedlich. In diesem und dem nächsten Kapitel werden Sie immer wieder weitere Funktionen kennen lernen. Vordefinierte Funktionen in PHP sind dann auch das alleinige Thema von Kapitel 6.

Durch folgende Zeile wird eine Konstante mit Namen `MAXWERT` definiert und auf den Wert 10 gesetzt.

```
define("MAXWERT", 10);
```

Um im Skript auf die Konstante zuzugreifen, schreiben Sie sie direkt ohne Dollarzeichen. Das ist auch der formale Unterschied zu den Variablen.

```
echo MAXWERT; /* gibt 10 aus */
```

Wenn Sie versuchen, einer Konstanten einen neuen Wert zuzuweisen, erhalten Sie eine Fehlermeldung.

Normalerweise spielt die Groß- und Kleinschreibung von Konstanten eine Rolle. Wenn diese hingegen nicht relevant sein soll, übergeben Sie einen dritten Parameter `true`:

```
define("MAXWERT", 10, true);
echo maxwert; /* gibt 10 aus */
```

Im Unterschied zu Variablen können Sie Konstanten nicht direkt in einem String in Anführungszeichen ausgeben lassen, da der PHP-Interpreter sie nicht von Text unterscheiden kann:

```
echo "Der maximale Wert ist MAXWERT"; /* Gibt aus: Der maximale Wert ist MAXWERT */
```

Über `define()` definieren Sie selbst Konstanten. Daneben stellt Ihnen PHP viele *vordefinierte Konstanten* zur Verfügung – z. B. mathematische Konstanten wie die Zahl π:

```
echo M_PI; /* 3.14159265359 */
```

Tipp

Mehr mathematische Konstanten finden Sie im PHP-Manual unter *http://www.php.net/manual/de/math.constants.php*.

Über eine weitere vordefinierte Konstante können Sie sich beispielsweise Informationen über die PHP-Version anzeigen lassen:

```
echo "Verwendete PHP-Version" . PHP_VERSION . "<br />\n";
```

So genannte *magische Konstanten* liefern Ihnen Informationen über das aktuelle Skript. Sie werden mit zwei Unterstrichen am Anfang und am Ende geschrieben. `__LINE__` liefert Ihnen die aktuelle Zeile des Skripts, `__FILE__` den Namen der Datei oder – ab PHP 5.3 – `__DIR__` den Namen des Ordners, in dem sich das Skript befindet:

Listing 4.10: Vordefinierte Konstanten (vordefinierte_konstanten.php)

```
01 echo "PI: ";
02 echo M_PI;
03 echo "<br />\n";;
04 echo "Verwendete PHP-Version: ";
05 echo PHP_VERSION;
06 echo "<br />\n";
07 echo "Aktuelle Zeile des Skripts: ";
08 echo __LINE__;
09 echo "<br />\n";
```

```
10 echo "Name der Datei: ";
11 echo __FILE__;
12 echo "<br />\n";
13 echo "Name des Ordners: ";
14 echo __DIR__;
15 echo "<br />\n";
```

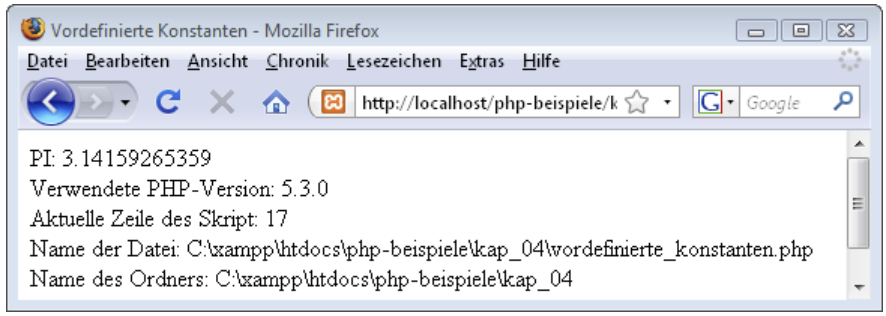


Abbildung 4.8: Ausgabe von Listing 4.10

Setzen Sie eine Konstante ein, die nicht definiert ist, so erhalten Sie bei entsprechendem Fehlermeldungsniveau einen Hinweis (Notice). PHP interpretiert diese aber ansonsten als String und gibt sie einfach direkt aus. Lassen Sie beispielsweise `__DIR__` in einer Version vor PHP 5.3 ausgeben, erhalten Sie die Notice und »__DIR__« wird ausgegeben.

Tipp

Alle vordefinierten Konstanten in PHP finden Sie unter *http://www.php.net/manual/en/reserved.constants.php*.

4.5 Operatoren

Operatoren brauchen Sie für Berechnungen und zur Verkettung von Zeichenketten.

4.5.1 Arithmetische Operatoren

Natürlich gibt es in PHP auch die in Programmiersprachen üblichen arithmetischen Operatoren. Die folgende Tabelle listet die fünf gebräuchlichen Operatoren für Zahlen auf:

Operator	Operation	Beispiel
+	Addition	`$i = 6 + 4; // 10`
-	Subtraktion	`$i = 6 - 4; // 2`
*	Multiplikation	`$i = 6 * 4; // 24`
/	Division	`$i = 6 / 4; // 1.5`
%	Modulo	`$i = 6 % 4; // 2`

Tabelle 4.2: Arithmetische Operatoren

Die meisten arithmetischen Operatoren kennen Sie sicher. Neu wird Ihnen aber eventuell der Modulo-Operator (`%`) sein, der den ganzzahligen Rest einer Division zurückgibt.

```
$i = 6 % 4;
```

Der Rest der Division von 6 durch 4 ist 2, so erhält `$i` den Wert 2. Mit dem Modulo-Operator lässt sich beispielsweise leicht ermitteln, ob eine Zahl gerade ist oder nicht. Denn wenn bei der Teilung durch 2 kein Rest übrig bleibt, ist die Zahl gerade.

```
$z = $i % 2; /* Wenn $z gleich 0, dann ist $i gerade */
```

Punkt vor Strich

Wenn Sie Berechnungen im PHP-Code durchführen, dann gilt, so wie man es erwarten würde, die Regel »Punkt vor Strich«. Das heißt, dass in einem Ausdruck wie

```
$i = 5 - 3 * 2;
```

zuerst die Multiplikation ausgeführt wird (`3 * 2`) und danach die Subtraktion. Deswegen erhält im obigen Beispiel `$i` den Wert `-1`. Wenn Sie hingegen wollen, dass zuerst eine andere Operation durchgeführt werden soll, müssen Sie Klammern einsetzen:

```
$k = (5 - 3) * 2;
```

Jetzt wird zuerst `5 - 3` berechnet und das Ergebnis mit `2` malgenommen, `$k` erhält also den Wert `4`.

Das sind die beiden wichtigsten Regeln zur Rangfolge der Operatoren. Weitere Regeln lesen Sie in Kapitel 5.

Kombinierte Operatoren

Häufig ändert man einen Wert und speichert den geänderten Wert wieder in der Variablen:

```
$i = 5;
$i = $i + 2;
```

`$i` hat jetzt den Wert `7`.

Tipp

An diesem Beispiel sehen Sie noch einmal deutlich, dass das »=«-Zeichen nicht bedeutet »ist gleich«, sondern »erhält den Wert«.

`$i = $i + 2;` lässt sich kürzer schreiben über einen so genannten kombinierten Operator +=:

```
$i = 5;
$i += 2;
```

Diese kombinierten Operatoren gibt es ebenfalls für die anderen Operatoren.

```
$i *= 2; /* entspricht $i = $i * 2; */
$i -= 2; /* entspricht $i = $i - 2; */
$i /= 2; /* entspricht $i = $i / 2; */
$i %= 2; /* entspricht $i = $i % 2; */
```

Sehr häufig möchte man einen Wert um eins erhöhen.

```
$i += 1;
```

Speziell hierfür gibt es noch einen weiteren Operator: den Inkrementoperator.

```
$i++;
```

Entsprechend verringert der Dekrementoperator den Wert um 1.

```
$i--;
```

4.5.2 Strings verknüpfen

Neben den Operatoren für Zahlen gibt es auch welche für Strings, also Texte. Am wichtigsten ist der Verknüpfungsoperator zur Verkettung von Strings – der Punkt:

```
$vorname = "Denis";
echo "Hallo " . " Welt. "; /* Hallo Welt */
echo "Hallo " . $vorname; /* Hallo Denis */
echo "<br />Guten Morgen " . $vorname . " - gut geschlafen?";
```

Wie Sie sehen, können Sie hierüber auch Variablen anketten.

Den Verknüpfungsoperator für Strings können Sie ebenfalls mit einer Zuweisung kombinieren über .=. Zuerst sehen Sie die ausführliche Variante:

```
$koffer = "Zahnbürste, ";
$koffer = $koffer . "Hose ";
$koffer = $koffer . "und T-Shirt";
echo "Ich nehme $koffer mit";
```

Es wird also die Variable `$koffer` mit einem Anfangswert belegt, der dann nach und nach mit weiteren Strings ergänzt wird. Die Ausgabe zeigt Abbildung 4.9.

Das lässt sich über .= auch verkürzen, wie in Listing 4.11.

Listing 4.11: Verknüpfungsoperator für Strings (verknuepfungsoperator.php)

```
$koffer = "Zahnbürste, ";
$koffer .= "Hose ";
$koffer .= "und T-Shirt";
echo "Ich nehme $koffer mit";
```

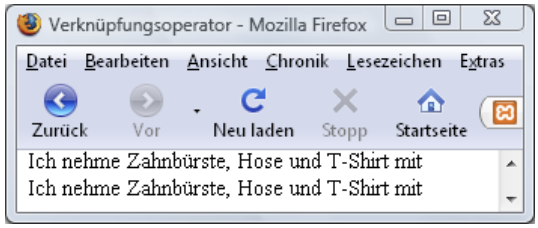


Abbildung 4.9: Die Ausgabe ist in beiden Fällen gleich.

4.6 Datentypen

Es gibt verschiedene Typen von Daten, mit denen PHP arbeiten kann, Strings und Zahlen haben Sie bereits kennen gelernt. Nun ist also der richtige Zeitpunkt, um diese und die weiteren möglichen Datentypen einmal genauer zu betrachten.

Die Datentypen werden in PHP – im Unterschied beispielsweise zu den streng typisierenden Sprachen wie Java – jedoch üblicherweise nicht vom Programmierer explizit gesetzt, sondern von PHP aus dem Kontext erkannt.

4.6.1 Strings

Den Typ String oder Zeichenkette haben Sie schon kennen gelernt. Ein String besteht aus ein oder mehreren Zeichen. Strings werden in einfachen oder doppelten Anführungszeichen notiert. Wahlweise können Sie auch die HereDoc- oder die NowDoc-Konstruktion benutzen (siehe Abschnitt 4.3.5):

```
$text  = "Das hier ist ein String";
$text2 = 'Das hier ist auch ein String';
$text3 = "7"; /* auch ein String */
$text4 = "10 Eier";
```

4.6.2 Integer und Float

Außerdem gibt es zwei unterschiedliche Typen für Zahlen: Integer für ganze Zahlen und Float für Fließkommazahlen.

Integer können positiv oder auch negativ sein und werden nicht in Anführungszeichen geschrieben.

```
$ganzezahl = 42;
$nocheine  = -13;
```

Integer werden sicher am häufigsten als Dezimalzahlen angegeben, das heißt mit 10 als Basis. Aber Sie können auch Zahlen definieren, die eine andere Basis als 10 haben, wie Oktalzahlen oder Hexadezimalzahlen. Bei Oktalzahlen, die als Basis 8 haben, wird eine `0` vorangestellt.

```
$oktal = 012; /* entspricht 10 */
```

Hexadezimalzahlen mit der Basis 16 kennen Sie von den Farbangaben in (X)HTML/CSS: In PHP schreiben Sie am Anfang von Hexadezimalzahlen `0x`:

```
$hexadezimal = 0xFF; /* entspricht dezimal 255 */
```

Fließkommazahlen (Float) werden mit einem Punkt geschrieben:

```
$float = 1.5;
```

Ebenfalls möglich ist die wissenschaftliche Schreibweise für Fließkommazahlen:

```
$a = 1.2e3; /* entspricht 1.2 * 10^3, d.h. 1200 */
$b = 7e-2; /* entspricht 7 * 10^-2 , d.h. 0.07 */
```

Hinweis

Neben *Float* finden Sie an manchen Stellen auch die Bezeichnung *Double*. Float und Double sind bei PHP identisch, der Name Double taucht mitunter aus historischen Gründen auf.

4.6.3 Wahrheitswerte

Der Boolesche Typ ist ein weitere möglicher Datentyp, dabei handelt es sich um einen Wahrheitswert. Er kann nur `true` (wahr) oder `false` (falsch) annehmen. Sie haben ihn schon als drittes Argument von `define()` kennen gelernt (Abschnitt 4.4).

```
$regnen = true;
```

Die Groß- und Kleinschreibung ist dabei nicht relevant, Sie können auch `TRUE` und `FALSE` schreiben. Boolesche Werte brauchen Sie bei der Überprüfung von Bedingungen: Wenn beispielsweise eine Meldung ausgegeben werden soll, wenn der Benutzer einen gültigen Benutzernamen eingegeben hat. Das Ergebnis einer Überprüfung ist dann `true` oder `false`. Mehr dazu in Kapitel 5.

4.6.4 Weitere Datentypen

Es gibt noch weitere so genannte zusammengesetzte Typen: Arrays und Objekte. Zu Arrays gleich mehr (Abschnitt 4.7), Genaueres zu Objekten lesen Sie in Kapitel 5.

Außerdem gibt es noch Ressourcen, die eine Referenz auf eine externe Ressource beinhalten, wie beispielsweise auf eine geöffnete Datei oder auf eine Verbindung zu einer Datenbank. Wie Sie mit PHP auf Dateien zugreifen, ist Thema von Kapitel 12, in Kapitel 11 geht es um Datenbankverbindungen.

`NULL` ist ein weiterer Datentyp und repräsentiert eine Variable ohne Wert. Das bedeutet: Diesem Typ gehört eine Variable an, der Sie noch keinen Wert zugewiesen haben, die Sie explizit auf `NULL` gesetzt haben oder die Sie mit der PHP-Funktion `unset()` gelöscht haben.

4.6.5 Immer der richtige Typ

Eine Variable kann innerhalb eines Skripts beliebig den Wert wechseln.

```
$a = "Hallo"; // String
$a = 7; // Integer
$a = 3.5; // Float
```

In diesem Beispiel ist `$a` zuerst vom Typ String, dann Integer und schließlich eine Fließkommazahl. PHP führt Konvertierungen zwischen den einzelnen Variablentypen automatisch durch. Es ermittelt automatisch den Typ einer Variable aus dem Kontext.

Float und Integer

Was das Ergebnis einer Berechnung ist – eine Fließkommazahl oder ein Integer – ist eigentlich so, wie man es intuitiv erwarten würde. Um das genauer anzusehen, brauchen wir eine Methode, um zu ermitteln, welchem Datentyp eine bestimmte Variable

angehört. Hier bietet sich die Funktion `var_dump()` an. `var_dump()` übergeben Sie in runden Klammern die Variable, über die Sie mehr Informationen erhalten möchten. `var_dump()` gibt dann den Inhalt der Variablen und den Typ aus.

Listing 4.12: Jonglieren zwischen Integer und Float (integer_float.php)

```
01 $a = 20;
02 $b = 3;
03 $c = 3.5;
04 $d = -3;
05 $e = -20;
06
07 $erg = $a / $b;
08 var_dump($erg);
09 echo "<br />n";
10 $erg2 = $a + $b;
11 var_dump($erg2);
12 echo "<br />\n";
13 $erg3 = $a + $c;
14 var_dump($erg3);
15 echo "<br />\n";
16 $erg4 =  $a / $e;
17 var_dump($erg4);
```

In den ersten fünf Zeilen werden Variablen mit Werten vorbelegt. Zeile 7 berechnet `20 / 3`. Das Ergebnis samt Variablentyp wird über `var_dump()` in Zeile 8 ausgegeben: Ein `float` mit dem Wert `6.666666667`. In Zeile 10 werden zwei ganze Zahlen (`20 + 3`) addiert, das Ergebnis ist ein Integer, wie man erwarten würde. Genauso einsichtig sind auch die beiden anderen Ergebnisse. Die Ausgabe des Skripts zeigt Abbildung 4.10.

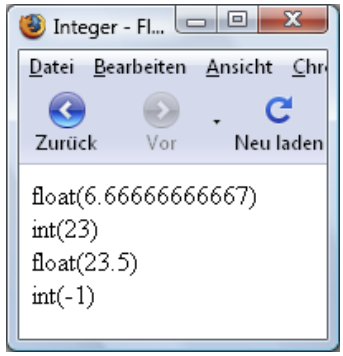


Abbildung 4.10: Das Ergebnis der Berechnungen mit Angabe des Dateityps

Konvertierung von String in Zahlen

Werden Strings und Zahlen kombiniert, findet die Konvertierung automatisch statt. Die Konvertierung von Zahl zu String ist einfach, aus `7` wird eben `"7"`. Umgekehrt gibt es teilweise ungewöhnliche Ergebnisse.

Um das an einem Beispiel zu zeigen, benötigen wir wieder einen Kontext, der eine Konvertierung von einem String in eine Zahl auslöst, beispielsweise die Addition. Bei der Konvertierung gilt folgendes Prinzip: Wenn ein String mit einer Zahl beginnt, wird diese genommen und der Rest verworfen. Wenn der String nicht mit einer Zahl beginnt, wird der String zu `0` konvertiert. Das gilt aber nur für das Ergebnis, der Inhalt der Variablen selbst bleibt unverändert. Ein Beispiel zeigt das:

Listing 4.13: Beispiel für die automatische Konvertierung von Strings (string_zu_zahl.php)

```
01 $str1 = "10 Eier";
02 $str2 = "Schachtel mit 10 Eiern";
03 $str3 = "3.5 Äpfel";
04 $erg1 = $str1 + 2;
05 var_dump($erg1);
06 echo "<br />\n";
07 $erg2 = $str2 + 2;
08 var_dump($erg2);
09 echo "<br />\n";
10 $erg3 = $str3 + 2;
11 var_dump($erg3);
```

In Zeile 4 wird `"10 Eier" + 2` berechnet. Das Ergebnis ist `12`. Das Ergebnis von `"Schachtel mit 10 Eiern" + 2` ist hingegen `2`. Denn `"Schachtel mit 10 Eiern"` beginnt nicht mit einer Zahl und wird als `0` ausgewertet. `"3.5 Äpfel" + 2` (Zeile 10) ergibt dann entsprechend 5.5 und ist ein Float. Die Ausgabe sehen Sie in Abbildung 4.11.

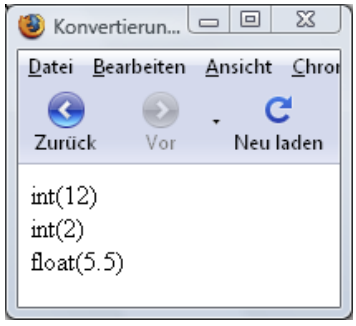


Abbildung 4.11: Das Ergebnis der Addition mit Strings und Zahlen

Schön sind diese Umwandlungen nicht und im Normalfall wird man vermeiden, so etwas zu tun. Wichtig ist aber für Sie: Falls Sie einen String mit einer Zahl addieren, wird das PHP klaglos durchführen, Sie erhalten keine Warnung oder Fehlermeldung.

4.7 Arrays

Die Typen von Variablen, die bisher besprochen wurden, speichern genau einen Wert. Manchmal möchte man aber gleichzeitig mit mehreren Werten arbeiten, beispielsweise mit einer Liste von möglichen Farben, einer Liste von Gästen, einer Liste von zur Verfügung stehenden Versionen, Sprachen, einer Liste von Preisen oder Produkten etc. Genau dafür sind Arrays gedacht, die mitunter auch Felder genannt werden.

Wenn man in einer Variablen mehrere Werte speichert, stehen viele nützliche Möglichkeiten offen: Die Werte lassen sich sortieren und neu ausgeben, man kann auf einzelne gezielt zugreifen, sie vergleichen, zählen, weitere ergänzen und wieder ausgeben lassen.

4.7.1 Arrays erstellen

Um ein Array zu erstellen, verwenden Sie das Schlüsselwort `array()`. Hier einmal ein Beispiel für ein einfaches Array mit drei Elementen:

```
$antworten = array("nie", "manchmal", "oft");
```

In Klammern hinter `array()` führen Sie bei der Definition eines Arrays die einzelnen Werte durch Komma getrennt auf. Wenn es Strings sind, schreiben Sie sie wie gewohnt in Anführungszeichen. Zahlen notieren Sie ohne:

```
$werte = array(42, 66, 3.5, 55, 7);
```

Innerhalb eines Arrays können auch verschiedene Typen kombiniert werden:

```
$antworten = array("nie", "manchmal", "oft", 42);
```

Die einzelnen Elemente werden von PHP automatisch durchnummeriert, die Nummerierung beginnt dabei – das ist wichtig – bei 0. Das ist der so genannte Index. Um ein einzelnes Element auszulesen, schreiben Sie den Namen des Arrays und in eckigen Klammern den Index.

```
echo $antworten[0]; /* nie */
echo "<br />\n";
echo $antworten[2]; /* oft */
```

Sie können Arrays auch problemlos im Nachhinein mit weiteren Elementen ergänzen.

Nehmen wir noch einmal das bestehende Array:

```
$antworten = array("nie", "manchmal", "oft", 42);
```

Dann können Sie durch folgende Zeile einfach ein weiteres Element anhängen:

```
$antworten[] = "aus Prinzip nicht";
```

Und das ließe sich natürlich ausgeben:

```
echo $antworten[4];
```

4.7.2 Informationen über Arrays ausgeben lassen

Wenn Sie versuchen, das Array als Ganzes per `echo` auszugeben, sieht das Ergebnis nicht wie gewünscht aus:

```
echo $antworten;
```

Das schreibt einfach »Array« auf den Bildschirm.

Um sich schnell einen Überblick über die Inhalte zu verschaffen, ist die PHP-Funktion `print_r()` praktisch.

Listing 4.14: Ausschnitt aus dem Listing arrays_print_r.php

```
print_r($antworten);
```

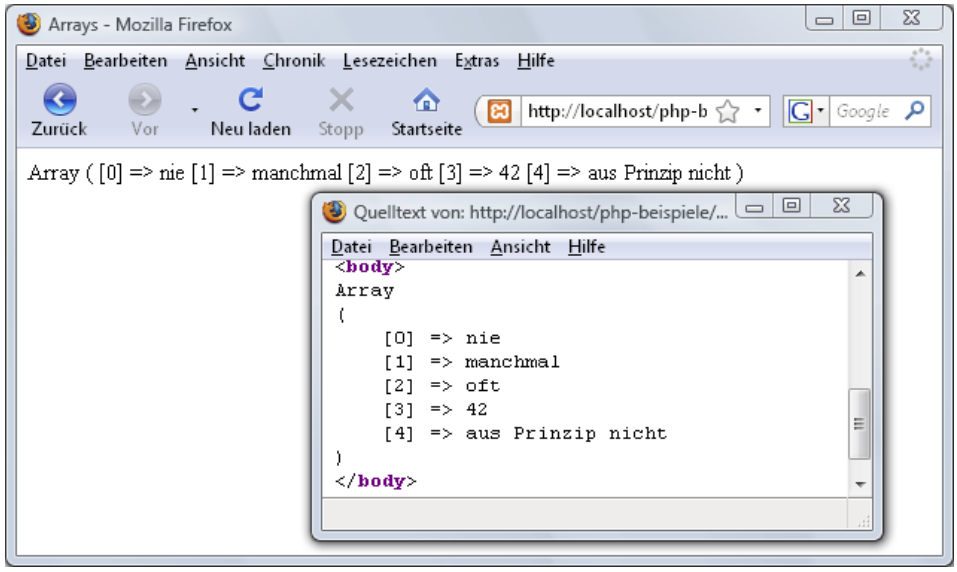


Abbildung 4.12: print_r() zeigt, was in Ihrem Array steckt.

Abbildung 4.12 zeigt das Ergebnis von `print_r()`: Die Anzeige der Arrayinhalte mit den zugehörigen Indizes wird im Quellcode noch übersichtlicher angezeigt. Diese Einrückung wird natürlich im Browser nicht dargestellt, da Einrückungen im (X)HTML-Quellcode vom Browser ignoriert werden.

Wollen Sie den Browser dazu bringen, die Inhalte wie im Quellcode anzuzeigen, inklusive aller Leerzeichen, können Sie das ansonsten selten verwendete (X)HTML-Element `pre` benutzen und die Ausgabe von `print_r()` innerhalb von Start- und End-tag von `pre` schreiben.

Listing 4.15: Mit ergänztem (X)HTML-Element `pre` (arrays_print_r_pre.php)

```
echo "<pre>";
print_r($antworten);
echo "</pre>";
```

Noch ausführlichere Informationen über Ihr Array erhalten Sie, wenn Sie anstelle von `print_r()` die Funktion `var_dump()` benutzen:

Listing 4.16: Der Inhalt des Arrays wird dieses Mal über die Funktion `var_dump()` ausgegeben (arrays_var_dump.php).

```
echo "<pre>";
var_dump($antworten);
echo "</pre>";
```

Sie sehen dann gleichzeitig, um welchen Datentyp es sich handelt, und bei Strings auch ihre Länge.

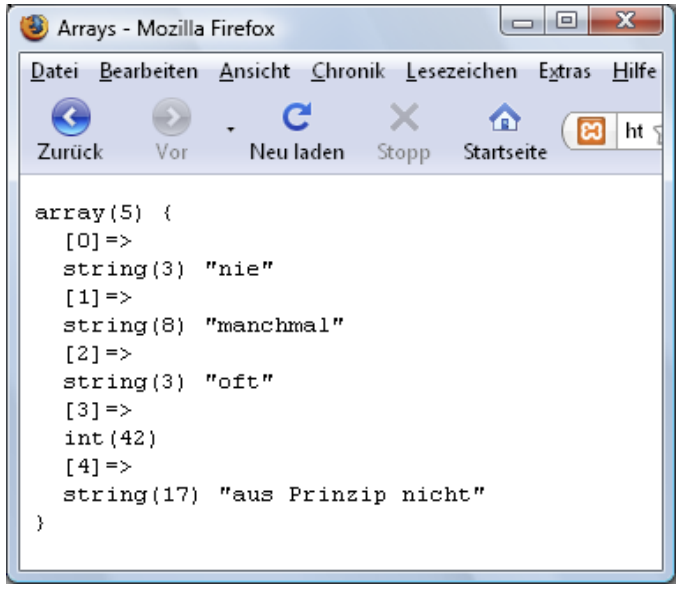


Abbildung 4.13: var_dump() liefert ausführlichere Informationen zu den Inhalten von Arrays.

4.7.3 Arrays durchlaufen mit foreach

Die Ausgabe mit `var_dump()` oder `print_r()` ist nur geeignet, um sich bei der Programmierung einen schnellen Überblick über den Inhalt zu verschaffen – man könnte diese Ausgabe nicht einem normalen Benutzer zumuten. Dafür gibt es andere Wege: Speziell für die Ausgabe oder sonstige Bearbeitung aller Elemente eines Arrays existiert die Schleife `foreach`. Bei `foreach` werden Schritt für Schritt die einzelnen Elemente des Arrays durchlaufen und die von Ihnen festgelegten Anweisungen für jedes ausgeführt. Sie müssen `foreach` nicht sagen, wie oft es das durchführen soll, denn `foreach` wird durch die Anzahl der Arrayelemente selbst begrenzt.

In runden Klammern hinter `foreach` geben Sie zuerst das Array an, das Sie durchlaufen möchten. Danach folgt das Schlüsselwort `as` und danach der Name einer Variablen, die den Wert der einzelnen Elemente zwischenspeichert. Dieser ist frei wählbar. In geschweiften Klammern steht der Code, der für jedes Element ausgeführt werden soll. Um jedes Element auszugeben, verwenden Sie den Namen, den Sie für die temporäre Variable eingesetzt haben.

Durch folgenden Code wird jedes Element des `$antworten`-Arrays ausgegeben – und danach jeweils ein Zeilenumbruch:

```
foreach ($antworten as $aw) {
  echo "$aw <br />\n";
}
```

Wenn Sie außerhalb von `foreach` noch einmal auf die Variable `$aw` zugreifen, erhalten Sie den zuletzt dort gespeicherten Array-Wert.

Listing 4.17: Arrays können über foreach durchlaufen werden (arrays_foreach.php).

```
foreach ($antworten as $aw) {
  echo "$aw <br />";
}
echo $aw; /* aus Prinzip nicht */
```

Um die Anzahl der Elemente eines Arrays zu ermitteln, können Sie die Funktion `count()` einsetzen. Bei `count()` notieren Sie in runden Klammern das Array, dessen Elemente Sie zählen möchten. Als Rückgabewert erhalten Sie die Anzahl der Elemente:

```
$anzahl = count($antworten);
echo $anzahl; // 5
```

4.7.4 Zufällig ein Bild anzeigen lassen

Jetzt ein kleines Beispiel für die Verwendung von Arrays. Es soll zufällig eines von mehreren Bildern ausgegeben werden. Die Pfade zu den Bildern werden dafür in einem Array gespeichert.

Außerdem benötigen wir eine Funktion, die eine zufällige Zahl ermittelt. Genau dafür gibt es `rand()`. `rand()` erwartet in runden Klammern zwei Werte: Der eine bestimmt den minimalen Wert der Zufallszahl, der andere gibt den höchsten möglichen Wert an:

```
$zufallszahl = rand(0, 4);
```

Damit wird eine Zahl von 0 bis einschließlich 4 in `$zufallszahl` gespeichert.

Jetzt zur zufälligen Ausgabe von Bildern.

Listing 4.18: Welches Bild angezeigt wird, bestimmt der Zufall (zufallsbilder.php).

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <title>Zufallsbilder</title>
06   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
07 </head>
08 <body>
09 <?php
10 $bilder = array("blumen.jpg", "boot.jpg",
11                 "landschaft.jpg", "stadt_am_meer.jpg",
12                 "strand.jpg");
13 $max = count($bilder) - 1;
14 $zufallszahl = rand(0, $max);
15 echo "<img src='$bilder[$zufallszahl]' height='200' width='150' />";
16 ?>
17 </body>
18 </html>
```

In Zeile 10 wird ein Array namens `$bilder` angelegt. Es beinhaltet die Pfade zu den Bildern, die sich im selben Ordner befinden wie das PHP-Skript selbst.

In Zeile 13 wird die Anzahl der Elemente des Arrays ermittelt und 1 davon abgezogen. Damit haben wir den höchsten Index des Arrays. Im Beispiel enthält das Array 5 Elemente. Der letzte Index aber – da beim Index mit 0 zu zählen begonnen wird – ist 4, also eins weniger.

Zeile 14 ruft die Funktion `rand()` auf. Sie soll eine Zahl zwischen 0 und dem in `$max` gespeicherten höchsten Index erzeugen. Diese wird in der Variablen `$zufallszahl` gespeichert.

In Zeile 15 erfolgt die Ausgabe des Zufallsbildes über das hierfür benötigte `img`-Element, das beim Attribut `src` den Pfad zur Datei erwartet. Hier wird auf das Array `$bilder` zurückgegriffen und als Index die Variable `$zufallszahl` benutzt, die ja einen Wert zwischen 0 und dem letzten Index enthält. Damit wird immer ein anderes Bild aus dem Bilderarray ausgelesen.

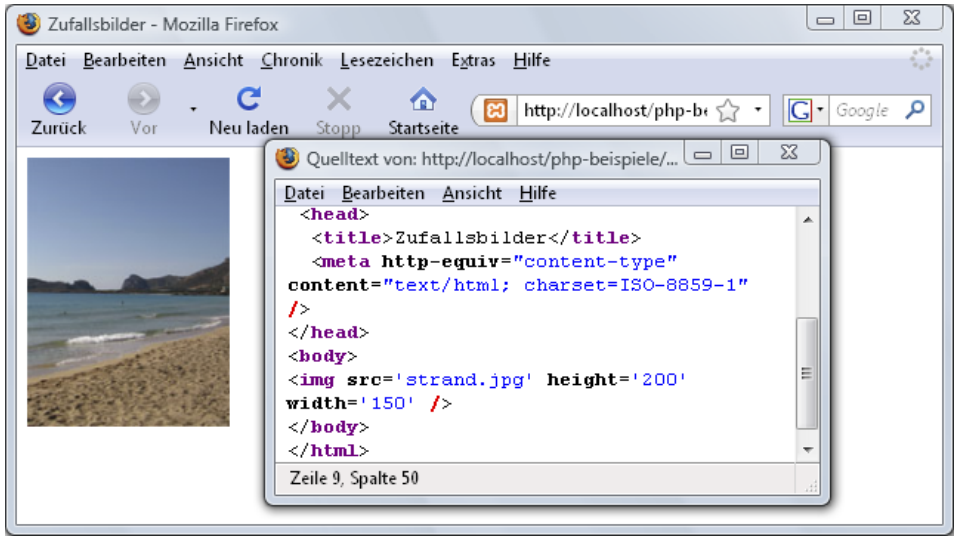


Abbildung 4.14: Zufallsbild – und anbei der erzeugte Quellcode

Wenn Sie das Skript testen, klicken Sie mehrmals auf den Reload-Button: Welche Bilder angezeigt werden, wird zufällig bestimmt.

Tipp

Das Beispiel sollte den Zusammenhang von Index und Anzahl der Elemente eines Arrays illustrieren. Sonst hätte man sich eine Zeile Code sparen können, indem man die von PHP zur Verfügung gestellte Funktion `array_rand()` benutzt, die aus dem Array, das man ihr in Klammern übergibt, zufällig einen Index wählt. Das Beispiel finden Sie unter dem Namen *zufallsbilder_array_rand.php* ebenfalls in den Listings, die Sie auf der Verlagswebsite zu diesem Buch unter *http://www.addison-wesley.de/9783827327239.html* herunterladen können.

4.7.5 Assoziative Arrays

Bisher haben wir die einzelnen Elemente über Nummern angesprochen. Manchmal möchte man aber die Arrayelemente über Namen ansprechen. Solche Schlüssel-Wert-Paare können Sie einsetzen, wenn Sie beispielsweise deutsche Farbnamen den entsprechenden in (X)HTML/CSS üblichen hexadezimalen Farbbezeichnungen zuordnen möchten. Oder um Vorwahlnummern Städten zuzuordnen, Produktklassen zu Mehrwertsteuersätzen usw. Auch das ist mit Arrays möglich. Diese Sorte von Arrays wird im Gegensatz zu den gerade besprochenen indizierten Arrays als *assoziative Arrays* bezeichnet.

Zur Erstellung eines assoziativen Arrays verwenden Sie wieder `array()`, schreiben aber in runde Klammern immer die Schlüssel-Wert-Paare, die durch `=>` verknüpft werden:

```
$farben = array ("rot"  => "#FF0000",
                 "grün" => "#00FF00",
                 "blau" => "#0000FF");
```

Wollen Sie die Funktion `array()` nicht einsetzen, können Sie die Elemente eines assoziativen Arrays auch einzeln definieren.

```
$farben["rot"]  = "#FF0000";
$farben["grün"] = "#00FF00";
```

Auf diese Art lassen sich auch nachträglich weitere Elemente ergänzen:

```
$farben["schwarz"] = "#000000";
```

Einzelne Werte sprechen Sie an, indem Sie in eckigen Klammern den Schlüssel schreiben:

```
echo $farben["rot"];
```

Hinweis

Es gibt auch viele in PHP vordefinierte assoziative Arrays. So können Sie über `$_SERVER["PHP_SELF"]` auf den Pfad zum aktuellen Skript zugreifen oder über `$_GET["name"]` oder `$_POST["name"]` auf den Inhalt von Formulardaten. Das ist Thema von Kapitel 7.

Einen schnellen Überblick über den Inhalt eines Arrays verschaffen Sie sich wiederum mit `print_r()` oder `var_dump()`.

```
print_r($farben);
```

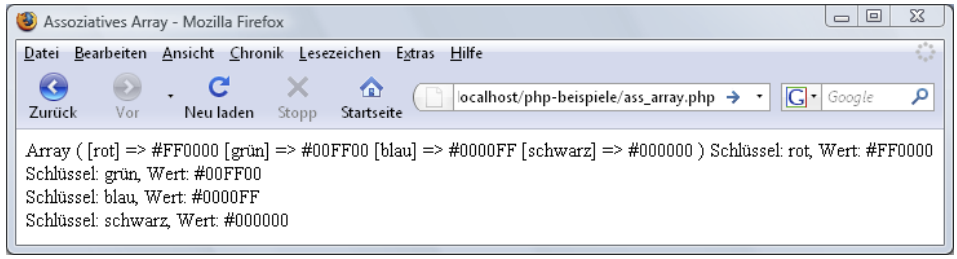


Abbildung 4.15: Das assoziative Array wird ausgegeben: oben über print_r(), unten über foreach.

Um die Inhalte ansprechender auszugeben, brauchen Sie `foreach`. In runden Klammern geben Sie zuerst den Namen des Arrays an, das durchlaufen werden soll. Dann folgen das Schlüsselwort `as` und zwei Variablen, die als temporäre Speicher für jeweils den Schlüssel und den dazugehörigen Wert dienen und durch `=>` getrennt werden.

Listing 4.19: Assoziative Arrays können ebenfalls über foreach ausgegeben werden (ass_array.php).

```
foreach ($farben as $k => $v){
  echo "Schlüssel: $k, Wert: $v<br />\n";
}
```

4.7.6 Schlüssel von Arrays richtig angeben

Noch einmal zur Ausgabe eines einzelnen Elements. Bei diesem schreiben Sie den Schlüssel in den eckigen Klammern in Anführungszeichen, sofern es sich um einen String handelt.

```
echo $farben["rot"];
```

Lassen Sie bei diesem Schlüssel, der ein String ist, die Anführungszeichen weg, erhalten Sie einen Hinweis (Notice). PHP beschwert sich, dass eine nicht definierte Konstante verwendet wird.

```
echo $farben[rot];
```

Abbildung 4.16: Ein Hinweis erscheint, wenn man bei einem Schlüssel, der ein String ist, keine Anführungszeichen setzt.

Hinweis

Sie erinnern sich? Konstanten werden ohne Dollarzeichen geschrieben. Bei nichtdefinierten Konstanten nimmt PHP an, dass es sich um den entsprechenden String handelt, und gibt diesen aus.

Diese Schreibweise ohne Anführungszeichen begegnet Ihnen mitunter noch in älteren Skripten, Sie sollten sie aber vermeiden. Es könnte zu Problemen führen, wenn Sie einmal in einem Skript wirklich eine Konstante mit demselben Namen – hier in unserem Fall `rot` – definiert hätten.

Handelt es sich hingegen beim Schlüssel um eine Zahl, verwenden Sie natürlich keine Anführungszeichen:

```
echo $antworten[0];
```

4.7.7 Arrays und Variableninterpolation

Nun zu Besonderheiten bei der Interpolation von Arrayvariablen in Strings. Wenn der Schlüssel eine Zahl ist, können Sie den Wert des Arrayelements problemlos in doppelten Anführungszeichen ausgeben lassen:

```
echo "Sag niemals $antwort[0]";
```

Wenn Sie hingegen einen String als Schlüssel haben, funktioniert das so schon einmal nicht:

```
echo "die Farbe ist $farben["rot"]"; /* geht nicht */
```

Auch mit einfachen Anführungszeichen geht es nicht:

```
echo "die Farbe ist $farben['rot']"; /* geht nicht */
```

Mit einfachen Anführungszeichen klappt es hingegen, wenn Sie – wie bereits in Abschnitt 4.3.4 vorgestellt – die geschweiften Klammern zur Klammerung des Ausdrucks verwenden:

```
echo "die Farbe ist {$farben['rot']}; /* geht */
```

Zwei weitere Varianten gibt es noch: Sie können den Verknüpfungsoperator einsetzen, um das Problem elegant zu umgehen:

```
echo "die Farbe ist " . $farben["rot"]; /* geht auch */
```

Es funktioniert außerdem noch, wenn Sie den Schlüssel ohne Anführungszeichen schreiben:

```
echo "die Farbe ist $farben[rot]"; /* geht auch */
```

4.7.8 Verschachtelte Arrays am Beispiel

Arrays können Sie auch verschachteln. Eben hatten wir ja ein Beispiel, in dem zufällig eins von mehreren Bildern angezeigt wurde. Dabei wurde ein `img`-Element ausgegeben mit unterschiedlichen Pfadangaben. Was aber, wenn man noch mehr Informa-

tionen zum jeweiligen Bild ausgeben lassen möchte? Obligatorisch wäre ja eigentlich das `alt`-Attribut für einen alternativen Text, außerdem könnte man das `img`-Element noch mit einem `title`-Attribut bestücken. Der Inhalt des `title`-Attributs wird von Browsern in Form eines Tooltipps angezeigt. Damit müsste beispielsweise folgender Code erzeugt werden:

```
<img src='stadt_am_meer.jpg' height='200' width='150' alt='Häuser' title='Griechische
Häuser am Abend' />
```

Dieses Mal soll sich also nicht nur der Inhalt des `src`-Attributs ändern, sondern es sollen entsprechend auch andere Texte für `alt` und `title` gezeigt werden.

Dafür braucht man ein verschachteltes Array.

Die Bildinformationen zu einem einzelnen Bild werden als assoziatives Array gespeichert:

```
array("pfad" => "stadt_am_meer.jpg",
     "alt"   => "Häuser",
     "titel" => "Griechische Häuser am Abend");
```

Entsprechend auch für die anderen Bilder. Aus diesen wird dann ein verschachteltes Array gebaut, also ein Array, das selbst wieder Arrays als Elemente hat. Im Beispiel heißt das Array `$bilder` und enthält als Elemente die Arrays mit den einzelnen Bildinformationen.

```
01 $bilder = array(
02               array("pfad"  => "blumen.jpg",
03                     "alt"   => "rote Blumen",
04                     "titel" => "Strauß aus roten Blumen"),
05               array("pfad"  => "landschaft.jpg",
06                     "alt"   => "Landschaft",
07                     "titel" => "Landschaft im Nebel"),
08               array("pfad"  => "stadt_am_meer.jpg",
09                     "alt"   => "Häuser",
10                     "titel" => "Griechische Häuser am Abend"),
11               array("pfad"  => "strand.jpg",
12                     "alt"   => "Strand",
13                     "titel" => "Strand mit Bergen"),
14               array("pfad"  => "boot.jpg",
15                     "alt"   => "Boot",
16                     "titel" => "Boot auf einem Felsen")
17          );
```

Um auf einzelne Werte zuzugreifen, schreiben Sie zuerst den Namen des Arrays, also `$bilder`. Dahinter folgen zwei eckige Klammernpaare. In die ersten schreiben Sie den Index des verschachtelten Arrays, auf das Sie zugreifen wollen, und in die zweiten eckigen Klammern den Namen des Werts, den Sie auslesen möchten:

```
echo $bilder[0]["pfad"]; // blumen.jpg
```

Damit lässt sich das Skript zur zufälligen Ausgabe von Bildern mit mehr Informationen folgendermaßen erstellen:

Listing 4.20: Dieses Mal können bei den einzelnen per Zufall angezeigten Bildern die jeweils passenden alt- und title-Werte bestimmt werden (zufallsbilder_erweitert.php).

```
/*Definition des verschachtelten Arrays wie oben */
18 $max = count($bilder) - 1;
19 $zufallszahl = rand(0, $max);
20 echo "<img src='{$bilder[$zufallszahl]['pfad']}'
21       height='200' width='150'
22       alt='{$bilder[$zufallszahl]['alt']}'
23       title='{$bilder[$zufallszahl]['titel']}' />\n";
```

Abbildung 4.17: Zufallsbilder mit jeweils angepassten title-Attributen, deren Inhalt im Browser als Tooltipp angezeigt wird.

PHP stellt viele nützliche Funktionen zur Arbeit mit Arrays bereit. Mehr dazu in Kapitel 6.

4.8 Nützlich für alle Zwecke: Dateien einbinden

Zum Abschluss des Kapitels geht es um eine praktische Funktion zum Einbinden von Dateien. Oft haben Sie bei Webprojekten Bereiche, die auf allen Webseiten vorkommen – beispielsweise einen Kopfbereich oder eine Fußzeile. Wenn sich der Inhalt bei einem dieser Bereiche ändert, müssen Sie jede Datei einzeln bearbeiten, wo dieser Bereich vorkommt. Praktischer ist es, diese Bereiche in einzelnen Dateien auszulagern und dann per PHP einzubinden.

Genau hierfür gibt es in PHP zwei Sprachkonstrukte, `include` und `require`. Sehen wir uns erst einmal die Funktionsweise von `include` an. An die Stelle, an der Sie die externe Datei einbinden wollen, notieren Sie `include` und dahinter den Pfad zur Datei, die Sie einbinden möchten.

Im folgenden Beispiel wird `include` zweimal eingesetzt: Am Anfang des Dokuments wird damit ein Begrüßungstext ausgegeben, am Ende des Dokuments wird über eine externe Datei ein Copyright-Vermerk ergänzt.

Listing 4.21: Zwei Dateien werden per include eingebunden (include_beispiel.php).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Dateien einbinden</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<?php
include "header.php";
?>
<h2>Lorem ipsum dolor </h2>
<p>sit amet ....</p>
<?php
include "copyright.php";
?>
</body>
</html>
```

Jetzt zu den eingebundenen Dateien. Der Inhalt von *copyright.php* ist ganz kurz, die Datei besteht nur aus einer Zeile:

Listing 4.22: Die Datei copyright.php ist einzeilig.

```
<p>&copy; 2009 Example.com</p>
```

In *copyright.php* steht nur (X)HTML-Code: ein Absatz mit einem ©-Zeichen, Jahreszahl und eine fiktive Domain.

Nun zur zweiten eingebundenen Datei: *header.php*. Diese beinhaltet hingegen PHP-Code: Hier wird ein Willkommensgruß mit dem aktuellen Datum ausgegeben.

Listing 4.23: Der Inhalt von header.php

```
<?php
date_default_timezone_set("Europe/Berlin");
echo "<h1>Willkommen am ";
```

```
echo date("j.n.Y");
echo "</h1>\n";
?>
```

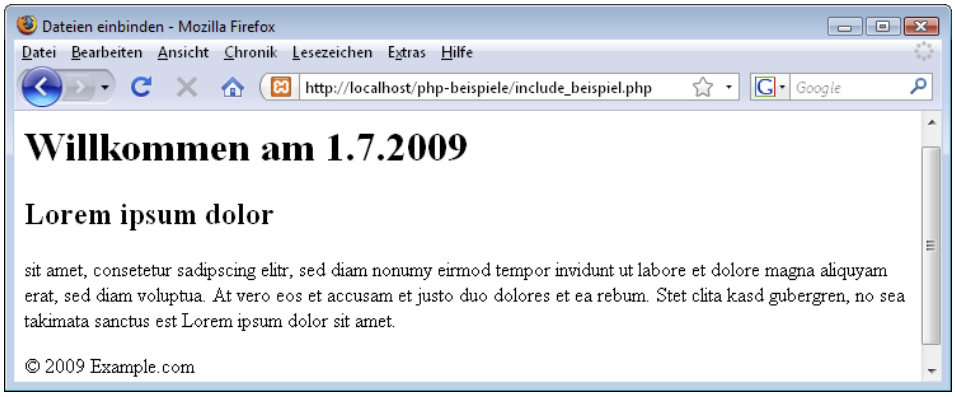


Abbildung 4.18: Die Ausgabe des Dokuments mit den zwei eingebundenen Dateien

Wie Sie gesehen haben, können Sie mit `include` sowohl Dateien einbinden, die nur (X)HTML-Code enthalten, als auch in den eingebundenen Dateien PHP-Befehle schreiben. Wenn Sie PHP-Code einbinden wollen, müssen Sie in der eingebundenen Datei dann aber den Code auch mit `<?php` einleiten und mit `?>` beenden, wie Sie in der Datei *header.php* sehen.

Neben `include` gibt es `require`, das prinzipiell genauso funktioniert:

```
require "header.php";
```

Der Unterschied zwischen `require` und `include` zeigt sich nur, wenn die angegebene Datei nicht geladen werden kann. In beiden Fällen wird eine Warnung ausgegeben, aber bei `require` zusätzlich noch ein fataler Fehler und die Abarbeitung des Skripts wird abgebrochen.

Wie bereits erwähnt, sollte die Ausgabe der Fehlermeldungen beim echten Einsatz der Skripte unterbunden werden. Und dann wird der Unterschied zwischen `include` und `require` sehr deutlich: Bei `include` wird der restliche Inhalt der Seite normal angezeigt, bei `require` hingegen nicht. Das heißt, `require` verwenden Sie zur Einbindung von essenziellem Code, ohne den der Rest der Verarbeitung nicht mehr sinnvoll ist. `include` benutzen Sie hingegen für Fälle wie im Beispiel. Hier wäre es sinnvoll, die Seite trotzdem ausgeben zu lassen, auch wenn z. B. die Copyright-Information fehlt.

Eine Einstellung, die für `include` und `require` relevant ist, ist der so genannte *include-path*. Dieser sagt dem Skript, wo es nach eingebundenen Dateien nachsehen soll. Standardmäßig sind hier ein Punkt und der Pfad zu PEAR angegeben. Worauf dieser gesetzt ist, sehen Sie wieder in der Ausgabe von `phpinfo()`.

include_path	.;C:\xampp\php\pear\	.;C:\xampp\php\pear\

Abbildung 4.19: Einstellung für den include_path bei XAMPP unter Windows

Bei XAMPP unter Windows steht: `.;C:\xampp\php\pear\`. Der Punkt am Anfang steht für das aktuelle Verzeichnis, danach kommt der Strichpunkt als Trennzeichen für die Angabe von mehreren Verzeichnissen und noch der Pfad `C:\xampp\php\pear`. Das bedeutet: Wird `include` oder `require` eingesetzt, wird zuerst nach der entsprechenden Datei ausgehend vom aktuellen Verzeichnis nachgesehen und sie dann, falls sie hier nicht gefunden wurde, im Verzeichnis *C:\xampp\php\pear* gesucht.

Unter Linux/Unix wird als Trennzeichen für mehrere Pfadangaben nicht der Strichpunkt, sondern der Doppelpunkt eingesetzt.

Tipp

Wenn Sie mit mehreren verschachtelten Includes in Unterverzeichnissen arbeiten, sollten Sie absolute Pfade verwenden. Benutzen Sie dann:

```
include __DIR__ . "/pfad/zur/include/datei";
```

Die Konstante `__DIR__` steht erst ab PHP 5.3 zur Verfügung, vorher mussten Sie Folgendes schreiben:

```
include dirname( __FILE__ ) . '/pfad/zur/include/datei;
```

5 Mehr Basics

Im letzten Kapitel haben Sie wichtige Sprachelemente wie Variablen und die Verwendung von unterschiedlichen Datentypen kennen gelernt. In diesem Kapitel geht es um weitere wichtige Grundlagen: Sie erfahren, wie Sie Ihre Skripte über Verzweigungen flexibler gestalten und über Schleifen Anweisungen mehrmals ausführen können. Außerdem geht es darum, wie sich über Funktionen und dann auch Objekte Code wiederverwenden lässt. Schließlich erhalten Sie noch Tipps zur Fehlersuche.

5.1 Je nachdem ... Entscheidungen fällen

Häufig läuft ein Programm nicht starr ab, sondern soll flexibel auf Bedingungen reagieren. Dafür braucht man Verzweigungen. Beispielsweise soll je nachdem, ob der Besucher im Formular auch seinen Namen eingegeben hat, etwas anderes geschehen: Wenn kein Name eingetragen ist, soll beispielsweise eine entsprechende Meldung ausgegeben werden, ansonsten soll er persönlich begrüßt werden. In Kapitel 7 bei den Formularen erfahren Sie, wie das im Detail geht. Wichtig aber hier: Für diese und andere Fälle brauchen Sie Verzweigungen.

5.1.1 if – elseif – else

Für Verzweigungen im Programmablauf dient `if`. Nach `if` steht in runden Klammern ein Ausdruck, der als Boolescher Ausdruck ausgewertet wird, und entweder `true` oder `false` ergibt. Ist `true` das Ergebnis und somit die Bedingung wahr, wird die Anweisung in geschweiften Klammern ausgeführt. Im folgenden Beispiel ist die Bedingung wahr, da 5 größer als 4 ist.

```
$i = 5;
if ($i > 4) {
  echo "$i ist größer als 4";
  /* hier können weitere Anweisungen folgen */
}
```

Für eine weitere Operation, die ausgeführt werden soll, wenn die Bedingung nicht zutrifft, verwenden Sie `else`:

```
$i = 5;
if ($i > 4) {
  echo "$i ist größer als 4";
  /* hier können weitere Anweisungen folgen */
} else {
  echo "$i ist nicht größer als 4";
  /* weitere Anweisungen bei Bedarf */
}
```

Bei `else` können Sie keine weitere Bedingung angeben – denn `else` umfasst einfach »alles Sonstige«. Wenn Sie weiter differenzieren wollen, d. h. mehrere Bedingungen testen, benutzen Sie `elseif`.

Listing 5.1: Verzweigungen (if_elseif_else.php)

```
$i = 5;
if ($i > 4) {
  echo "$i ist größer als 4";
} elseif ($i == 4) {
  echo "$i gleich 4";
} else {
  echo "$i ist kleiner als 4";
}
```

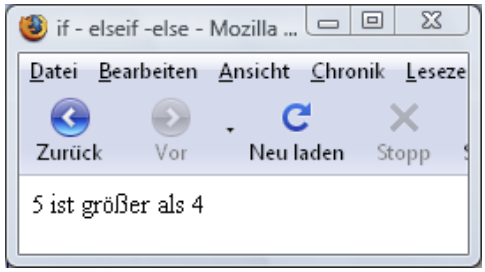


Abbildung 5.1: Das Ergebnis der Verzweigung mit if-elseif-else

Tipp

Übrigens können Sie auch innerhalb eines Anweisungsblocks den PHP-Modus verlassen, müssen danach aber wieder dorthin wechseln, um die schließende geschweifte Klammer zu ergänzen. Im folgenden Beispiel wird mehrmals zwischen PHP- und HTML-Modus gewechselt. Dies ist sinnvoll, wenn man längeren (X)HTML-Code ausgeben lassen möchte:

Listing 5.2: Munterer Wechsel zwischen PHP- und (X)HTML-Modus (html_php_wechsel.php)

```
01 <?php
02 $i = 5;
03 if ($i > 4) {
04 ?>
05 <p>Die Bedingung ist wahr.</p>
06 <?php
07 } else {
08 ?>
09 <p>Die Bedingung ist nicht wahr.</p>
10 <?php
11 }
12 ?>
```

Beachten Sie, dass in Zeile 10 noch einmal in den PHP-Modus gewechselt wird für die schließende Klammer in Zeile 11.

Doppeltes Ist-Gleich-Zeichen für Vergleiche

In den Beispielen wurden zwei Vergleichsoperatoren eingesetzt: > »größer als« und == zur Überprüfung, ob die Variable den Wert 4 hat. Zur Überprüfung, ob eine Variable einen bestimmten Wert hat, dient also ein *doppeltes Ist-Gleich-Zeichen*. Das einfache = hingegen weist einer Variablen einen Wert zu.

Und diese Unterscheidung ist wichtig. Falls Sie nämlich bei der Überprüfung das einfache = verwenden, erhalten Sie nicht das gewünschte Ergebnis:

Listing 5.3: Zuweisung anstelle von Vergleich (zuweisung_statt_vergleich.php)

```
$i = 5;
echo "Vor if ist \$i $i<br />\n";
if ($i = 4) {
  echo "\$i gleich 4<br />\n";
}
echo "Nach if ist \$i $i<br />\n";
```

In diesem Fall erhalten Sie keinerlei Fehlermeldung. Es wurde einfach die Zuweisung durchgeführt, der Wert in runden Klammern nach dem `if` wird als `4` ausgewertet und das entspricht dem Booleschen Wert `true`.

Konvertierung von und in Boolesche Werte

In Kapitel 4 haben Sie Beispiele gesehen, wie Strings in bestimmten Kontexten wie bei der Addition in Zahlen konvertiert werden. Genauso gibt es auch Konvertierungen zwischen Zahlen und Strings und Booleschen Werten. Die Bedingung in runden Klammern beim `if` ist ein Kontext, bei dem der Inhalt als Boolescher Wert ausgewertet wird.

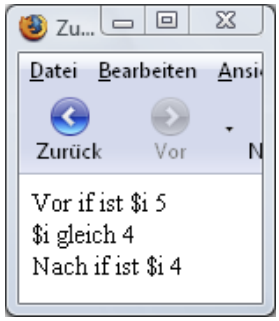


Abbildung 5.2: Dieses Ergebnis war sicher nicht beabsichtigt.

Folgende Werte werden als `false` behandelt:

- die Zahlen `0` und `0.0`
- eine leere Zeichenkette `""` oder auch eine Zeichenkette mit dem Element `"0"`
- ein Array ohne Elemente
- der spezielle Typ `NULL`
- und das Wort `false` selbst.

Alle anderen Ausdrücke werden zu `true` ausgewertet.

Wird ein Boolescher Wert per `echo` oder `print` ausgegeben, wird er dafür in einen String verwandelt. `false` entspricht einem leeren String, `true` der `1`.

Vergleichsoperatoren

Zwei Vergleichsoperatoren haben Sie im Beispiel eben gesehen: das > (»größer als«) und das ==. Es gibt natürlich weitere. Eine vollständige Liste finden Sie in der Tabelle 5.1.

<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
==	gleich
!= oder <>	Ungleich
===	Identität
!==	Nichtidentität

Tabelle 5.1: Vergleichsoperatoren

Neben < (»kleiner als«) und > (»größer als«), gibt es auch <= (»kleiner oder gleich«) und entsprechend >= (»größer oder kleiner«). Und das Gegenteil der Überprüfung auf

Gleichheit == ist != die Ungleichheit. Erklärungsbedürftig ist sicher der dreifache ===-Operator. Dieser überprüft nicht nur, ob ein Wert mit einem anderen gleich ist, sondern auch, ob er mit ihm identisch ist, das heißt, er berücksichtigt auch noch den Variablentyp.

Dreifaches Ist-Gleich-Zeichen

Ein Beispiel zum dreifachen Ist-Gleich-Zeichen. In Listing 5.4 wird `$i` der Wert `"4"` zugewiesen (Zeile 1), ist also ein String. Dann folgen zwei `if`-Anweisungen. In der ersten (Zeile 2) wird nur auf Gleichheit überprüft. PHP führt automatisch eine Typumwandlung durch und so ist die erste Prüfung wahr. In der zweiten Prüfung (Zeile 6) wird der Identitätsoperator eingesetzt, diese Prüfung schlägt fehl, da `$i` ein String ist und 4 ein Integer, sie also nicht vom selben Typ sind.

Listing 5.4: Bei === muss auch der Variablentyp übereinstimmen (identitaetsoperator.php).

```
01 $i= "4";
02 if ($i == 4) {
03   var_dump($i);
04   echo " == 4<br />\n";
05 }
06 if ($i === 4) {
07   var_dump($i);
08   echo " === 4<br />\n";
09 } elseif ($i !== 4) {
10   var_dump($i);
11   echo " !== 4<br />\n";
12 }
```

Im Beispiel wird außerdem noch immer über `var_dump()` der Typ ermittelt. In der Ausgabe in Abbildung 5.3 sehen Sie schön: Der String `"4"` ist gleich mit der Zahl `4`, aber nicht typidentisch mit `4`.

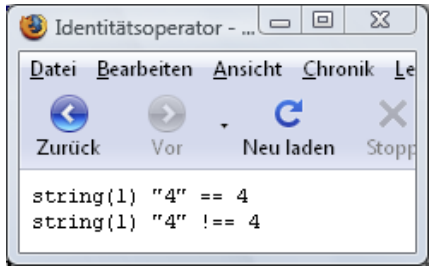


Abbildung 5.3: Ausgabe von Listing 5.4

Abkürzungen

Noch einmal zurück zur Syntax. Sie haben gesehen, dass der Anweisungsblock, also das, was ausgeführt werden soll, wenn eine Bedingung wahr ist, immer in geschweiften Klammern steht. Die geschweiften Klammern sind nur notwendig, wenn mehr als eine Anweisung ausgeführt werden soll; d. h. in den obigen Beispielen wären sie teilweise nicht notwendig. Es ist aber praktisch, sie zu setzen, da sie nicht falsch sind, die Lesbarkeit des PHP-Codes erhöhen und außerdem problemlos um weitere Anweisungen ergänzt werden können.

Eine weitere Verkürzungsmöglichkeit gibt es durch den ?-Operator. Als Beispiel eine einfache `if-else`-Anweisung:

```
$i = 6;
if ($i % 2 == 0) {
  $fazit = "gerade";
} else {
  $fazit = "nicht gerade";
}
echo "Das Ergebnis: $fazit";
```

Dies lässt sich über den ?-Operator verkürzen:

Listing 5.5: Verkürzung durch den ?-Operator (ternary_operator.php)

```
$fazit = ($i % 2 == 0) ? "gerade" : "ungerade";
echo "Das Ergebnis: $fazit";
```

Den ?-Operator setzen Sie ein, wenn Sie einer Variable, abhängig von der Auswertung einer Bedingung, einen anderen Wert zuweisen möchten. Hierfür notieren Sie den Ausdruck, der ausgewertet werden soll, und dann das Fragezeichen. Dahinter folgt zuerst der Ausdruck, der zugewiesen werden soll, wenn die Bedingung wahr ist, und nach einem Doppelpunkt der Ausdruck, der zugewiesen werden soll, wenn die Bedingung falsch ist.

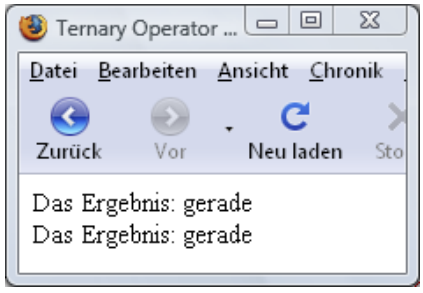


Abbildung 5.4: Das Ergebnis ist dasselbe – ob Sie if-else einsetzen oder den ?-Operator.

Eine weitere Verkürzung ist ab PHP 5.3 über den Operator `?:` möglich (*Ternary Short Cut*). Dabei wird der Ausdruck, der vor `?:` steht, überprüft: Ist er wahr, wird er zurückgeliefert, ansonsten der Ausdruck, der nach `?:` steht.

Listing 5.6: Ternary Short Cut (ternary_shortcut.php)

```
echo "<pre>";
var_dump(false ?: "Hallo");
var_dump(true ?: "Hallo");
$erg = (5 ?: 4);
echo $erg;
echo "</pre>";
```

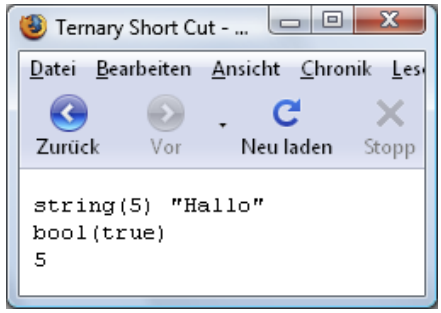


Abbildung 5.5: Die Ausgabe von Listing 5.6

5.1.2 Bedingungen kombinieren

Bis jetzt haben wir immer nur eine Bedingung getestet. Es lassen sich auch mehrere Bedingungen kombinieren:

Wenn zwei Bedingungen zutreffen sollen, verknüpfen Sie sie mit `&&`:

```
if (($i < 5) && ($j == 10)) { }
```

Der Ausdruck in Klammern hinter `if` wird als wahr ausgewertet, wenn gleichzeitig `$i` kleiner als 5 ist und `$j` den Wert 10 hat.

Soll hingegen nur eine von mehreren Bedingungen zutreffen, so verwenden Sie `||`:

```
if (($i < 5) || ($j == 10)) { }
```

Die Anweisung wird ausgeführt, wenn `$i` kleiner als 5, oder aber wenn `$j` den Wert 10 hat.

Ein Beispiel hierzu. Das folgende Skript gibt je nach Uhrzeit eine andere Begrüßung aus und der Seitenhintergrund wird andersfarbig eingefärbt:

Listing 5.7: Unterschiedliche Begrüßung je nach Tageszeit (untersch_begr.php)

```
01 <?php
02   date_default_timezone_set("Europe/Berlin");
03   $uhrzeit = date("H");
04   if ($uhrzeit < 5 || $uhrzeit > 20) {
05      $gruss = "Gute Nacht";
06      $hg = "#886CEA";
07   } elseif ($uhrzeit < 11) {
08      $gruss = "Guten Morgen";
09      $hg = "#66EDF0";
10   } elseif ($uhrzeit < 15) {
11      $gruss = "Guten Mittag";
12      $hg = "#D1F066";
13   } elseif ($uhrzeit < 18) {
14      $gruss = "Guten Nachmittag";
15      $hg = "#F09366";
16   } else  {
17      $gruss = "Guten Abend";
18      $hg = "#F066DF";
19   }
20 ?>
21 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
22     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
23 <html xmlns="http://www.w3.org/1999/xhtml">
24  <head>
25   <title>Unterschiedliche Begrüßung </title>
26   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
27 <style type="text/css">
28   body { background-color: <?php echo $hg; ?>; }
29 </style>
30 </head>
31 <body>
32 <?php
33   echo $gruss;
34 ?>
35 </body>
36 </html>
```

In Zeile 2 wird die Zeitzone gesetzt und in Zeile 3 die Uhrzeit ermittelt (Genaueres zur Funktion `date()` in Kapitel 6). Ab Zeile 4 gehen die `if-elseif-else`-Verzweigungen los. In Zeile 4 wird überprüft, ob die Uhrzeit kleiner als 5 oder größer als 20 ist. In diesem Fall wird die Variable `$gruss` mit dem Wert `"Gute Nacht"` belegt und in einer weiteren Variable `$hg` eine passende Hintergrundfarbe gespeichert.

Zeile 7 prüft, ob die Uhrzeit kleiner als 11 ist und weist wieder den Variablen `$gruss` und `$hg` passende Werte zu usw. In Zeile 28 wird dann die Hintergrundfarbe, die in `$hg` gespeichert ist, per CSS `body` zugewiesen. In Zeile 33 wird hingegen auf die andere Variable, `$gruss`, zugegriffen und der Gruß ausgegeben.

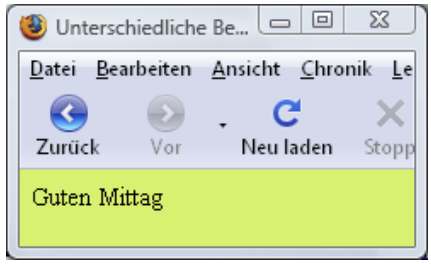


Abbildung 5.6: Begrüßungstext und immer andere Hintergrundfarbe

Diese Aufgabe lässt sich natürlich auf ganz verschiedene Arten lösen. Aber noch zu einem Punkt der vorgestellten Lösung. Angenommen, es ist 4 Uhr morgens. Dann trifft die erste Bedingung zu (Zeile 5), die `$uhrzeit` ist kleiner als 5. Es treffen aber an sich auch die `elseif`-Bedingungen in Zeile 9, 13 oder 17 zu, denn 4 ist auch kleiner als 11 oder 15. Das spielt jedoch hier keine Rolle, denn bei einer `if-elseif-else`-Konstruktion wird, wenn die erste Bedingung zutrifft, der entsprechende Code ausgeführt und mit der Abarbeitung des Skripts unterhalb des `if-elseif-else`-Blocks weitergemacht.

In Zeile 5 wurden zwei Bedingungen über `||` verknüpft, das heißt, dass nur eine von beiden wahr sein muss, damit der entsprechende Codeblock ausgeführt wird. Für `||` können Sie auch das englische Wort `OR` schreiben und anstelle von `&&` auch das englische Wort `AND`. Die Tabelle fasst es noch einmal zusammen.

Symbol	engl. Bezeichnung	Bedeutung
`$a && $b`	`$a AND $b`	Beide Bedingungen müssen wahr sein.
`$a || $b`	`$a OR $b`	Eine von beiden Bedingungen muss zutreffen.
	`$a XOR $b`	Eine der beiden – jedoch nicht beide – Bedingungen muss wahr sein.
`!$a`		Logische Negation, kehrt die Wahrheitswerte um.

Tabelle 5.2: Bedingungen verknüpfen

Rangfolge der Operatoren

Es gibt jedoch einen Unterschied zwischen `AND` und `&&` – der derselbe ist, der auch zwischen `OR` und `||` existiert. Sie unterscheiden sich in ihrem Rang. Der Rang von Operatoren spezifiziert, in welcher Reihenfolge mehrere Operatoren ausgewertet werden. Sie erinnern sich, dass auch in PHP die Punkt-vor-Strich-Regel gilt:

```
$i = 5 + 2 * 3;
```

Das Ergebnis dieser Rechnung ist 11 – da `2 * 3` zuerst ausgeführt wird und dann die Addition. Wie die Rangfolge von Operatoren ist, wird üblicherweise als Tabelle angegeben. Je weiter ein Operator oben steht, desto höheren Rang hat er und desto eher wird er ausgeführt.

`* / %` stehen in der Tabelle der Rangfolge oberhalb von `+ - .` (Addition, Subtraktion, Operator zur Verknüpfung von Strings), d. h. sie werden vor diesen ausgeführt.

Und genau in der Rangfolge unterscheiden sich auch `AND` und `&&`. `&&` hat einen höheren Rang als `AND`. Zwischen beiden befindet sich der Zuweisungsoperator (`=` etc.). Das heißt `&&` binden enger als die Zuweisungen, `AND` schwächer.

Bei der Auswertung von komplexen Ausdrücken spielt außerdem noch die Assoziativität eine Rolle. Ist die Rangfolge der Operatoren gleich, sagt die Assoziativität, in welche Richtung ausgewertet wird. Ein Beispiel:

```
$i = 5 - 7 - 8;
```

Das Minus-Zeichen hat eine linke Assoziativität, das heißt, dass der Ausdruck von links nach rechts ausgewertet wird. Das Ergebnis ist also `-10`.

Die folgende Tabelle listet für die bisher besprochenen Operatoren die Rangfolge und die Assoziativität auf: Desto weiter oben, desto eher ausgeführt.

Operator	Assoziativität
`! ++ --`	rechts
`* / %`	links
`+ - .`	links
`< <= > >=`	keine Richtung
`== != === !==`	keine Richtung
`&&`	links
`||`	links
`= += -= *= /= .= %=`	rechts
`and`	links
`xor`	links
`or`	links

Tabelle 5.3: Rangfolge der Operatoren

Hinweis

Die vollständige Tabelle mit der Rangfolge aller Operatoren finden Sie im PHP-Manual unter *http://www.php.net/manual/de/language.operators.php*.

5.1.3 switch

Wenn Sie mehrere einzelne Überprüfungen haben, so bietet sich eine alternative Konstruktion an: `switch`.

Schreiben Sie hinter `switch` in runden Klammern die Variable, die überprüft werden soll. Dann folgen in geschweiften Klammern die Überprüfungen. Für die einzelnen Fälle notieren Sie `case` und dahinter den möglichen Wert, auf den Sie überprüfen wollen. Nach einem Doppelpunkt folgen die Anweisungen, die in diesem Fall ausgeführt werden sollen und ein `break`. Am Ende können Sie noch `default` notieren für eine Anweisung, die ausgeführt werden soll, wenn keine der anderen Fälle zutreffen.

Listing 5.8: Mehrere Fälle abdecken mit switch (switch.php)

```
01 $dasda = "Apfel";
02 switch ($dasda) {
03   case "Apfel":
04     echo "$das da ist ein Obst";
05     break;
06   case "Karotte":
07     echo "$das da ist ein Gemüse";
08     break;
09   case "Käse":
10     echo "$das da ist ein Milchprodukt";
11     break;
12   default:
13     echo "Kenne ich nicht";
14 }
```

Die Ausgabe ist, wie Sie wahrscheinlich erwarten (Abbildung 5.7).

Abbildung 5.7: Ausgabe der switch-Anweisung

Im Beispiel sind es Strings (`"Apfel"`, `"Karotte"`), auf die geprüft wird. Wenn Sie auf numerische Werte prüfen würden, würden Sie diese ohne Anführungszeichen schreiben:

```
case 7:
  echo "... ist 7";
```

Jetzt zur Bedeutung von `break`. Sehen wir uns einmal an, was passiert, wenn wir es weglassen:

Listing 5.9: Hier steht kein break bei den einzelnen Fällen (switch_ohne_break.php).

```
01 $dasda = "Apfel";
02 switch ($dasda) {
03   case "Apfel":
04     echo "$das da ist ein Obst<br />";
05   case "Karotte":
06     echo "$das da ist ein Gemüse<br />";
07   case "Käse":
08     echo "$das da ist ein Milchprodukt<br />";
09   default:
10     echo "Kenne ich nicht<br />";
11 }
```

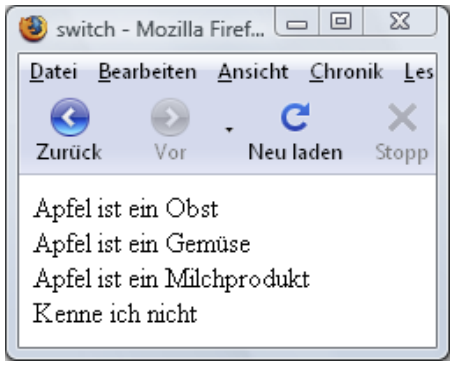


Abbildung 5.8: Das war wahrscheinlich nicht beabsichtigt ...

Abbildung 5.8 zeigt es deutlich: Ohne `break` werden nach dem ersten zutreffenden Fall alle folgenden Anweisungen ausgeführt. Das ist meist nicht erwünscht und genau das verhindert `break`.

5.2 Schleifen – mehrmals dasselbe tun

Mit Schleifen weisen Sie Ihr Programm an, etwas mehrmals zu tun. Eine Schleife haben Sie schon kennen gelernt: In Kapitel 4 haben Sie gesehen, wie man `foreach` einsetzt, um Arrays zu durchlaufen. Hier geht es um weitere Schleifenkonstruktionen: `for`, `while` und `do-while`.

5.2.1 while-Schleife

Beginnen wir mit der `while`-Schleife. Hinter `while` schreiben Sie in runden Klammern eine Bedingung, die überprüft werden soll. Solange die Bedingung erfüllt ist, wird die in geschweiften Klammern stehende Anweisung ausgeführt.

Das folgende Beispiel schreibt fünf Mal »hallo« auf den Bildschirm:

Listing 5.10: Eine einfache while-Schleife (while.php)

```
01 $i = 1;
02 while ($i < 6){
03   echo "hallo ";
04   $i++;
05 }
```

In Zeile 1 wird `$i` mit dem Wert `1` vorbelegt. In Zeile 2 beginnt die `while`-Schleife. In runden Klammern wird überprüft, ob `$i < 6` ist. Das ist der Fall. Die Anweisungen in den geschweiften Klammern werden ausgeführt: In Zeile 3 wird der Wert von `$i` und `hallo` ausgegeben. In der folgenden Zeile (4) wird `$i` um 1 erhöht (inkrementiert). War `$i` am Anfang 1, hat es jetzt den Wert 2.

Nun springt die Ausführung wieder zurück in Zeile 2 und die Bedingung wird erneut überprüft: Ist `$i` kleiner als `6`? Und die Anweisung wird erneut ausgeführt. Das geht so weiter, bis `$i` den Wert `6` erhält. Wieder wird die Bedingung geprüft, aber da `6` nicht kleiner als `6` ist, wird unterhalb der schließenden geschweiften Klammer mit der Abarbeitung des Skripts fortgefahren.

Je nachdem, welche Bedingung bei der `while`-Schleife steht, kann es auch sein, dass eine `while`-Schleife kein einziges Mal ausgeführt wird. Wenn beispielsweise `$i` im Listing zu Beginn den Wert `6` erhält, wird die `while`-Schleife nicht ausgeführt:

```
$i = 6;
while ($i < 6){
  echo "hallo";
  $i++;
}
```

5.2.2 do-while-Schleife: zumindest einmal

Genau das ist bei der `do-while`-Schleife anders. Diese wird *mindestens einmal* ausgeführt, da die Bedingung erst nach der ersten Ausführung steht.

Listing 5.11: Mindestens einmal ausgeführt: die do-while-Schleife (do_while.php)

```
$i = 6;
do {
  echo "hallo ";
```

```
  $i++;
}
while ($i < 6);
```

Das heißt, obwohl die Bedingung von Anfang an nicht zutrifft, wird die Schleife trotzdem einmal ausgeführt und einmal »hallo« ausgegeben.

5.2.3 Kompakt: die for-Schleife

Eine weitere Schleife ist die `for`-Schleife. Hinter `for` kommt eine runde Klammer mit drei Angaben: dem Anfangswert eines Zählers, der Bedingung und der Wertänderung. Diese drei Elemente werden durch Strichpunkt voneinander getrennt.

Listing 5.12: Die kompakte for-Schleife (for.php)

```
for ($i = 1; $i < 6; $i++){
  echo "$i. Hallo<br />\n";
}
```

Eine `for`-Schleife funktioniert so: Der Anfangswert wird gesetzt (`$i = 1`), dann die Bedingung überprüft (ist `$i` kleiner als 6?). Jetzt wird der Schleifenkörper abgearbeitet, hier der Wert von `$i` und `Hallo` mit einem Zeilenumbruch ausgegeben. Dann wird die Variable `$i` um eins erhöht. Und wiederum die Bedingung kontrolliert und eine neue Zeile geschrieben ... Dies geht solange, bis die Bedingung (`$i < 6`) nicht mehr wahr ergibt. Dann wird die Schleife beendet. Das Ergebnis ist dasselbe wie bei der `while`-Schleife.

Prinzipiell ist die `while`-Schleife besser geeignet als die `for`-Schleife, wenn die Anzahl an Ausführungen nicht feststeht. So wird beispielsweise eine `while`-Schleife eingesetzt, wenn man Daten aus einer Datenbanktabelle ausliest. Dabei verwendet man eine Funktion, die `false` zurückgibt, wenn keine Daten mehr vorhanden sind. Eine `for`-Schleife hingegen ist die richtige Wahl, wenn die Anzahl, die die Anweisungen durchgeführt werden sollen, von Vornherein schon feststeht. In unserem Beispiel ist deswegen die `for`-Schleife praktischer.

Bei `for`-, `while`- und `do-while`-Schleifen müssen Sie aufpassen, dass Sie keine Endlosschleife erstellen. Eine Endlosschleife würde beispielsweise durch folgenden Code erzeugt:

```
for ($i = 1; $i < 6; $i--){
  echo "$i. Hallo<br />\n";
}
```

Hier wird `$i` zu Beginn auf 1 gesetzt und überprüft, ob `$i` kleiner als `6` ist. Der Schleifenkörper wird abgearbeitet. Dann wird jedoch `$i` um eins verringert (dekrementiert). `$i` erhält also den Wert `0`. Da `$i` immer kleiner wird, wird die Bedingung (`$i < 6`) immer zutreffen und die Schleife theoretisch endlos laufen.

Tipp

Endlosschleifen können allerdings in der Kombination mit `break` oder `continue` durchaus sinnvoll sein (siehe hierzu Abschnitt 5.2.5).

5.2.4 Verschachtelte Schleifen

Schleifen können auch ineinander verschachtelt werden. Möchte man beispielsweise das kleine Einmaleins anzeigen lassen, braucht man zwei ineinander verschachtelte Schleifen. Im Beispiel soll das kleine Einmaleins als Tabelle ausgegeben werden. Bevor man daran geht, eine solche Aufgabe zu lösen, empfiehlt es sich, erst einmal zu überlegen, wie der erzeugte (X)HTML-Code aussehen soll. Sehen Sie sich bei Bedarf in Kapitel 3 noch einmal an, wie Tabellen in (X)HTML erstellt werden.

Nun zum Beispiel. Das kleine Einmaleins als Tabelle bedeutet, dass 10 Zeilen benötigt werden mit jeweils 10 Zellen. Dafür werden zwei `for`-Schleifen eingesetzt, die äußere ist für die Zeilen (`tr`) zuständig, die innere für die 10 Zellen (`td`) jeweils in einer Zeile.

Listing 5.13: Das kleine Einmaleins über PHP ausgegeben (einmaleins.php)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Einmaleins</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<table border="1" cellpadding="5">
<?php
for ($i = 1; $i <= 10; $i++) {
  echo "<tr>\n";
  for ($j = 1; $j <= 10; $j++) {
    $zahl = $i * $j;
    echo "\t<td>$zahl</td>\n";
  }
  echo "</tr>\n";
}
?>
</table>
</body>
</html>
```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Abbildung 5.9: Das kleine Einmaleins als Tabelle

In Zeile 9 beginnt die Tabelle. Dieser Code steht noch außerhalb von PHP. In Zeile 11 beginnt die äußere `for`-Schleife, die von 1 bis 10 hochzählt, und in Zeile 12 das `tr`-Starttag und in Zeile 17 das `tr`-Endtag ausgibt. Beim ersten Durchlauf hat `$i` den Wert `1`. Es wird `<tr>` geschrieben und in die innere Schleife gewechselt (Zeile 13).

Die innere `for`-Schleife verwendet als Zähler `$j`, da dieser unabhängig sein muss vom Zähler der äußeren Schleife. In der inneren Schleife wird das Produkt der beiden Zähler berechnet (Zeile 14) und jeweils in eine Zelle geschrieben (Zeile 15). Beim ersten Durchgang der inneren Schleife wird also `1*1` in die erste Zelle geschrieben, beim zweiten Durchgang `1*2`, beim dritten Durchgang `1*3` bis zu `1*10`. Sind die 10 Zellen erstellt, ist die innere Schleife beendet und `</tr>` wird ausgegeben.

Dann beginnt die äußere Schleife erneut. `$i` hat jetzt den Wert 2. Eine neue `<tr>` wird geschrieben und wieder in die innere Schleife gewechselt, die wieder 10 Zellen ausgibt, dieses Mal berechnet sie aber `2*1`, `2*2`, `2*3` usw. usf.

Die möglichen Verschachtelungen gehen natürlich weiter. Möchte man bei der ausgegebenen Tabelle jede zweite Zeile anders einfärben, braucht man noch eine `if`-Verzweigung. Um jede zweite Zeile anders einzufärben, wird jede zweite `tr` mit einer `class` versehen, die per CSS eine Hintergrundfarbe erhält (siehe auch Kapitel 3).

Listing 5.14: Jede gerade Zeile erhält eine CSS-Klasse, die für eine Hintergrundfarbe sorgt (einmaleins_farbig.php).

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <title>Einmaleins</title>
06   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
07 <style type="text/css">
08 .gerade { background-color: #FFDFBF; }
09 </style>
10 </head>
11 <body>
12 <table border="1" cellpadding="5">
13 <?php
14 for ($i = 1; $i <= 10; $i++) {
15   if ($i % 2 == 0) {
16     echo "<tr class='gerade'>\n";
17   } else {
18     echo "<tr>\n";
19   }
20   for ($j = 1; $j <= 10; $j++) {
21     $zahl = $i * $j;
22     echo "\t<td>$zahl</td>\n";
23   }
24   echo "</tr>\n";
25 }
26 ?>
27 </table>
28 </body>
29 </html>
```

Die Änderungen sind in Listing 5.14 fett markiert. In den Zeilen 7–9 wird per CSS für die Klasse `.gerade` eine Hintergrundfarbe definiert. In Zeile 15 wird geprüft, ob `$i` gerade ist und `$i` speichert ja die Nummer der aktuellen Zeile. Wenn die Zeilennummer gerade ist, dann bleibt kein Rest, wenn man die Zeilennummer durch 2 teilt, d. h. `$i % 2` ist 0. In diesem Fall wird das `tr`-Starttag geschrieben, in diesem aber noch `class='gerade'` ergänzt. Im anderen Fall (Zeile 18), wenn `$i` also nicht gerade ist, wird `tr` ohne Klassenangabe geschrieben.

5.2.5 Schleifen steuern über break und continue

`break` haben Sie bereits beim `switch`-Statement gesehen. Dort dient es dazu, die `switch`-Anweisung zu verlassen. Sie können `break` in dieser Funktion auch in anderen Schleifen einsetzen, also bei `for`, `while` oder `do-while`. Auch `continue` dient zur Steuerung von Schleifen. Hiermit bewirken Sie aber, dass wieder zum Anfang des Schleifenblocks zurückgesprungen wird.

Ein Beispiel zeigt `break` und `continue` im Einsatz:

Listing 5.15: Schleife steuern über break und continue (break_continue.php)

```
01 for ($i = 1; $i <= 10; $i++) {
02   if ($i == 3) {
03     continue;
04   }
05   echo "$i. Durchlauf<br />\n";
06   if ($i == 5) {
07     break;
08   }
09 }
```

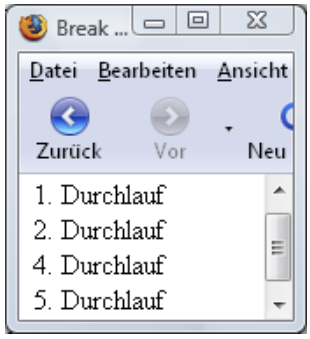


Abbildung 5.10: Durch den Einsatz von break oder continue wird die Schleife nicht von 1 bis 10 durchlaufen.

Im Listing sehen Sie eine `for`-Schleife, die an sich von 1 bis 10 durchläuft. Ausgegeben wird außerdem die Nummer des aktuellen Durchlaufs. Jetzt gibt es zwei `if`-Anweisungen. Die erste in Zeile 2 überprüft, ob `$i` den Wert `3` hat (achten Sie auf das doppelte Gleichheitszeichen). Ist das der Fall, wird mit `continue` bewirkt, dass die Ausführung wieder an den Anfang des Schleifenblocks zurückspringt. Das heißt, »3. Durchlauf« wird nicht ausgegeben. Eine weitere `if`-Anweisung in Zeile 6 prüft, ob `$i` den Wert `5` hat, dann wird mit `break` die `for`-Schleife verlassen. Deswegen wird nach »5. Durchlauf« nichts mehr ausgegeben. Das Ergebnis zeigt Abbildung 5.10.

Wenn Sie `break` oder `continue` ohne weitere Angabe schreiben, wird damit immer die aktuelle Schleife verlassen. Bei verschachtelten Schleifen können Sie aber auch festlegen, welche der Schleifen verlassen werden soll. Notieren Sie dabei einfach die Zahl der Schleife, aus der `break` ausbrechen soll bzw. bei `continue` zu deren Anfang zurückgesprungen werden soll.

Um das zu demonstrieren, brauchen wir eine Konstruktion mit verschachtelten Schleifen. Im folgenden Beispiel sehen Sie zwei ineinander verschachtelte `for`-Schlei-

fen. Innerhalb der äußeren Schleife wird der Zähler der äußeren Schleife und innerhalb der inneren Schleife der Zähler der inneren Schleife ausgegeben. Zuerst wird `break` ohne weitere Angabe benutzt.

Listing 5.16: break in einer verschachtelten Schleife (break_verschachelt.php)

```
01 for ($i = 1; $i <= 3; $i++) {
02   echo "außen: $i<br />\n";
03   for ($j = 1; $j <= 3; $j++) {
04     echo "innen $j<br />\n";
05     if ($j == 2)  {
06       break;
07     }
08   }
09 }
```

Durch das `break` in der inneren Schleife wird dafür gesorgt, dass diese nur bis 2 läuft. Statt `break` könnten Sie hier auch `break 1` schreiben. Die äußere Schleife hingegen läuft ungestört.

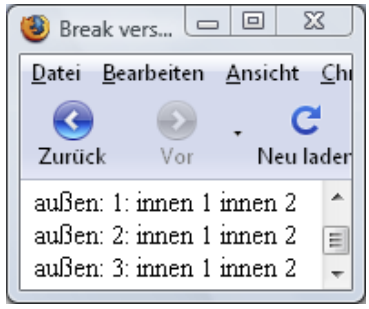


Abbildung 5.11: break bewirkt, dass die innere Schleife nur bis 2 läuft.

Ändern wir jetzt einmal das `break` in ein `break 2` – dann wird in diesem Moment nicht die innere, sondern die äußere Schleife verlassen:

Listing 5.17: Mit break die äußere Schleife verlassen (break_verschachtelt_2.php)

```
01 for ($i = 1; $i <= 3; $i++) {
02    echo "<br />außen: $i: ";
03   for ($j = 1; $j <= 3; $j++) {
04     echo "innen $j ";
05     if ($j == 2)  {
06       break 2;
07     }
08   }
09 }
```

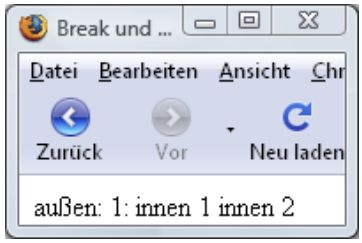


Abbildung 5.12: Jetzt wird die äußere Schleife durch break 2 verlassen.

5.2.6 goto

Neu in PHP 5.3 ist `goto`. Hinter `goto` geben Sie einen Bezeichner an, zu diesem wird dann direkt gesprungen:

Listing 5.18: Sprungmarken über goto (goto.php)

```
goto a;
echo "Grundsätzlich bin ich überflüssig";
a:
echo "Das wars schon";
```

`goto a` bewirkt, dass das Skript beim Bezeichner `a:` weitergeführt wird. Deswegen wird nur der zweite Text ausgegeben.

Exkurs: Alternative Syntax für Verzweigungen und Schleifen

Für `if-else` und Schleifen mit `while`, `for`, `foreach` und `switch` gibt es eine alternative Syntax. Anstelle der öffnenden geschweiften Klammer wird ein Doppelpunkt benutzt und anstelle der schließenden Klammer `endif`, `endwhile`, `endfor`, `endforeach` oder `endswitch`. Hier ein Beispiel für eine `if`-Verzweigung und eine `while`-Schleife

Listing 5.19: Verzweigungen und Schleifen können auch ohne geschweifte Klammern geschrieben werden (alternative_syntax.php).

```
$i = 5;
if ($i > 4):
  echo "$i ist größer als 4<br />\n ";
endif;
$j = 1;
while ($j < 6):
  echo "$j: hallo<br />\n";
  $j++;
endwhile;
```

5.3 Funktionen schreiben

PHP stellt Ihnen viele nützliche Funktionen zur Verfügung. Ein paar haben Sie bereits kennen gelernt wie beispielsweise `count()`, um die Elemente eines Arrays zu zählen, oder `rand()`, das Ihnen eine Zufallszahl zurückgibt, oder auch `var_dump()` zur Ausgabe von Informationen über Variablen.

Sie können aber auch selbst Funktionen definieren. Das ist nützlich für Codefragmente, die eine klare Aufgabe haben und häufiger vorkommen. Sie brauchen sie nur einmal zu schreiben, können sie jedoch beliebig oft aufrufen. Dadurch ist der Code besser organisiert und leichter auf Fehler zu durchsuchen.

Eine Funktion besteht aus dem Wort `function` und einem Namen, den Sie selbst bestimmen. (Zu den namenlosen Funktionen gleich in Abschnitt 5.3.4.) Danach stehen runde Klammern mit den Argumenten und in geschweiften Klammern folgt der Funktionsrumpf mit den Anweisungen.

```
function ausgabe()
{
  echo "Hallo München";
}
```

Um die Funktion auszuführen, müssen Sie sie aufrufen:

```
ausgabe();
```

Eine Funktion kann auch einen Wert zurückgeben. Dafür notieren Sie in der Funktion das Schlüsselwort `return` und dahinter den Wert, der zurückgegeben werden soll.

```
function copyright()
{
  return "Copyright 2009";
}
```

Wenn Sie die Funktion jetzt aufrufen, so können Sie den Wert, den die Funktion zurückgibt, einer Variablen zuweisen. Und mit dieser Variable können Sie dann weitere Operationen durchführen. Im Beispiel wird der Text in Großbuchstaben ausgegeben – das macht die Funktion `strtoupper()`.

```
$c = copyright();
echo strtoupper($c);
```

Übrigens beendet `return` die Funktion. Steht dahinter noch eine Anweisung, so wird diese nicht ausgeführt, im Beispiel hat echo also keinerlei Auswirkung:

Listing 5.20: Beispiele für Funktionen (funktionen.php)

```
function copyright2()
{
  return "Copyright 2009";
  echo "Hallo, hallo, warum hört mich denn niemand";
}
```

Tipp

Wollen Sie über `return` mehrere Werte zurückgeben, so können Sie ein Array zurückgeben lassen.

Die bisherigen Funktionen machen unter allen Umständen dasselbe. Flexibler werden Funktionen durch Parameter. Nehmen wir an, Sie haben eine Funktion, die Ihnen den Bruttowert berechnet. Dann möchten Sie wahrscheinlich den Bruttowert von *unterschiedlichen Beträgen* ermitteln. Dafür definieren Sie, dass Ihre Funktion einen Parameter erwartet. Hierfür schreiben Sie eine Variable in die runden Klammern bei der Funktionsdefinition. Mit dieser Variablen führen Sie dann im Funktionsrumpf die Berechnung durch.

```
function brutto($netto)
{
  return $netto * 1.19;
}
$betrag = 25;
$bruttowert = brutto($betrag);
echo "$betrag ergibt $bruttowert inkl. MWSt<br />\n";
```

Mehrere Parameter geben Sie durch Komma getrennt an. Im folgenden Beispiel wird die Funktion `brutto()` erweitert: Sie nimmt neben dem Nettobetrag auch den Mehrwertsteuersatz entgegen.

Listing 5.21: Funktion mit Parameter (funktionen_parameter.php)

```
function brutto2($netto, $mwstSatz)
{
  return $netto * (100 + $mwstSatz) / 100;
}
$betrag = 25;
$bruttowert = brutto2($betrag, 7);
echo "$betrag ergibt $bruttowert inkl. MWSt<br />\n";

$bruttowert2 = brutto2($betrag, 19);
echo "$betrag ergibt $bruttowert2 inkl. MWSt<br />\n";
```

5.3.1 Übergabe per Wert und per Referenz

Wenn Sie in einer Funktion einen Wert übergeben und diesen in der Funktion verändern, so verändert er sich dadurch nicht. Man spricht hier davon, dass einer Funktion eine Variable *als Wert* übergeben wird. Anders ist es, wenn Sie eine Variable *als Referenz* übergeben, dann können Sie innerhalb der Funktion den Wert verändern und das wirkt sich auch außerhalb der Funktion aus. Um einer Funktion eine Variable per Referenz zu übergeben, schreiben Sie bei der Parameterliste vor die Variable ein `&`-Zeichen, das ist der Referenzoperator.

```
function veraendern2(&$a, &$b) {}
```

Hierzu ein Beispiel. Im folgenden Listing gibt es zwei Funktionen. Diese erwarten jeweils zwei Parameter. Der ersten Funktion werden die Variablen als Werte übergeben, der zweiten als Referenz.

Listing 5.22: Unterschiede bei der Übergabe per Referenz und per Wert (referenz_wert.php)

```
01 echo "<h3>Übergabe per Wert</h3>\n";
02 function veraendern($a) {
03   $a++;
04 }
05 $c = 2;
06 echo "Vor Funktionsaufruf: \$c ist $c<br />\n";
07 veraendern($c);
08 echo "Nach Funktionsaufruf: \$c ist $c<br />\n";
09 echo "<h3>Übergabe per Referenz</h3>\n";
10 function veraendern2(&$a) {
11   $a++;
12 }
13 echo "Vor Funktionsaufruf: \$c ist $c<br />\n";
14 veraendern2($c);
15 echo "Nach Funktionsaufruf: \$c ist $c<br />\n";
```

In Zeile 2 wird die Funktion mit dem Namen `veraendern()` definiert, die einen Parameter erwartet. Diese werden dann im Funktionsrumpf verändert – `$a` wird inkrementiert.

In Zeile 5 wird die Variable `$c` definiert und mit dem Wert `2` belegt. Zeile 6 gibt den Wert der Variablen aus. In Zeile 7 wird die Funktion `veraendern()` aufgerufen und ihr die Variable übergeben. Zeile 8 gibt aus, welchen Wert die Variable nach dem Funktionsaufruf hat. Sie sehen es in Abbildung 5.13: Die Variable hat vor und nach dem Funktionsaufruf denselben Wert. Das ist so bei der standardmäßigen Übergabe per Wert.

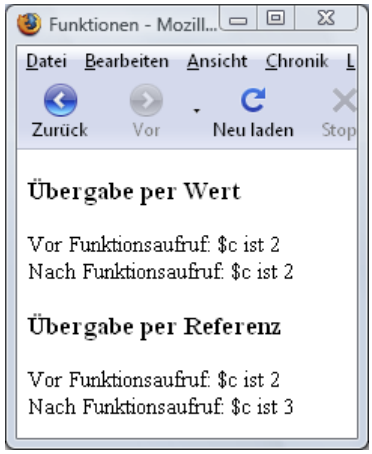


Abbildung 5.13: Durch die Übergabe der Variablen per Referenz wirkt sich die Veränderung innerhalb der Funktion auch außerhalb der Funktion aus.

Anders ist das Ergebnis der zweiten Funktion. In Zeile 10 sehen Sie eine weitere definierte Funktion. Hier wird die Variable per Referenz übergeben und vor dem Parameter steht der Referenzoperator (`&`). Ansonsten ist alles wie gehabt. Die Änderungen zeigen sich dann in der Testausgabe: Vor Funktionsaufruf hat die Variable den Wert 2. Nach Funktionsaufruf ist jedoch der Wert verändert: `$c` ist 3.

Hinweis

Wenn Sie hingegen *bei Aufruf der Funktion* die Variable per Referenz übergeben

```
veraendern(&$c);
```

so erhalten Sie ab PHP 5.3 einen Hinweis des Typs DEPRECATED: Der Hinweis informiert Sie, dass diese Art, Referenzen per Laufzeit zu übergeben, nicht mehr erwünscht ist (da er in PHP 6 nicht unterstützt werden wird) und dass Sie stattdessen die Funktion selbst modifizieren sollen – aber das Skript wird trotzdem ausgeführt.

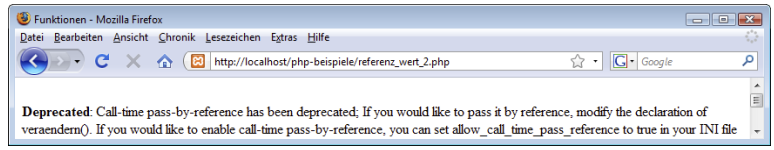


Abbildung 5.14: Übergabe der Variablen per Referenz beim Funktionsaufruf ist nicht mehr erwünscht.

5.3.2 Defaultwerte für Parameter

Bei Funktionen können Sie auch Defaultwerte für Parameter angeben, die genommen werden, wenn der entsprechende Parameter nicht angegeben ist. Nehmen wir als Beispiel noch einmal den Bruttoberechner. Wir können ihn jetzt so modifzieren, dass im Normalfall von einem Mehrwertsteuersatz von 19% ausgegangen wird.

```
function brutto ($netto, $mwstSatz = 19) { }
```

Die Funktion können Sie nun auf zwei unterschiedliche Arten nutzen: Wenn Sie beim Aufruf nur einen Parameter angeben, wird der Defaultwert mit 19% Mehrwertsteuer benutzt. Soll ein anderer Mehrwertsteuersatz verwendet werden, so übergeben Sie der Funktion zwei Parameter:

Listing 5.23: Funktion mit Defaultwert (brutto_default_wert.php)

```
function brutto ($netto, $mwstSatz = 19)
{
  return $netto * (100 + $mwstSatz) / 100;
}
$betrag = 25;
$bruttowert = brutto ($betrag);
echo "$betrag ergibt $bruttowert inkl. MWSt<br />\n";
$bruttowert2 = brutto ($betrag, 7);
echo "$betrag ergibt $bruttowert2 inkl. MWSt<br />\n";
```

Wichtig ist dabei, dass zuerst die Parameter ohne Defaultangabe stehen und danach die Parameter mit Defaultwerten. Die umgekehrte Reihenfolge wird nicht funktionieren:

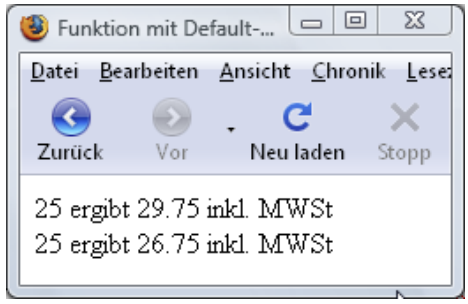


Abbildung 5.15: Ausgabe von Listing 5.23

Tipp

Übrigens verwendet PHP selbst bei vielen der vordefinierten Funktionen Defaultparameter. Erinnern Sie sich an die Funktion `define()` zur Definition einer Konstanten (Kapitel 4)? Hier geben Sie mindestens zwei Parameter an: zuerst den Namen der Konstanten, die Sie definieren möchten, und danach den Wert. Fakultativ können Sie hingegen noch als dritten Parameter `true` angeben, wenn Groß- und Kleinschreibung nicht beachtet werden soll.

5.3.3 Zugriff auf Variablen innerhalb und außerhalb von Funktionen

Eine Variable, die Sie außerhalb einer Funktion definieren, ist im globalen Geltungsbereich. Variablen innerhalb von selbst definierten Funktionen befinden sich hingegen im lokalen *Funktionsskopus* (Skopus = Geltungsbereich). Konkret heißt das: Sie können normalerweise nicht in Funktionen auf Variablen außerhalb der Funktion zugreifen.

Im folgenden Beispiel wird eine Variable `$vorname` definiert und ihr der String `"Lilli"` zugewiesen. In der Funktion wird versucht, auf diese Variable zuzugreifen und sie auszugeben.

```
$vorname = "Lilli";
function gruss()
{
  echo "Hallo $vorname<br />\n";
}
gruss();
```

Dies funktioniert jedoch nicht, das Skript gibt nur »Hallo« aus, und bei strengem Fehlermeldungslevel auch den Hinweis, dass eine undefinierte Variable benutzt wird.

Abbildung 5.16: Innerhalb einer Funktion können Sie nicht auf eine Variable, die außerhalb der Funktion definiert ist, zugreifen.

Wenn Sie innerhalb von Funktionen auf eine Variable außerhalb zugreifen wollen, brauchen Sie zusätzlich das Schlüsselwort `global`. Dieses schreiben Sie innerhalb der Funktion vor die Variable und kennzeichnen damit, dass Sie auf die gleichnamige Variable aus dem globalen Geltungsbereich zugreifen möchten.

Listing 5.24: Mit global greifen Sie auf außerhalb von Funktionen definierte Variablen zu (variablen_geltungsbereich.php).

```
$vorname = "Lilli";
function gruss()
{
  global $vorname;
  echo "Hallo $vorname<br />\n";
}
gruss();
```

Eine andere Möglichkeit ist der Einsatz des von PHP vordefinierten Arrays `$GLOBALS`. Dieses vordefinierte Array enthält als Elemente alle im globalen Geltungsbereich existierenden Variablen. Es ist ein assoziatives Array: Der Schlüssel ist jeweils der Name der Variablen, der Wert der Wert der Variablen.

Das bedeutet, über `$GLOBALS["vorname"]` können Sie innerhalb der Funktion auf den Inhalt von `$vorname` zugreifen.

Listing 5.25: Zugriff per $GLOBALS (variablen_geltungsbereich_2.php)

```
$vorname = "Lilli";
function gruss()
{
  echo "Hallo {$GLOBALS['vorname']}<br />\n";
}
gruss();
```

5.3.4 Lambda-Funktionen und Closures

Sie können auch Funktionen ohne Namen definieren, so genannte anonyme Funktionen, die auch Lambda-Funktionen genannt werden. Dieses fortgeschrittene, aus der funktionalen Programmierung entliehene Feature gibt es seit PHP 5.3.

Diese anonymen Funktionen können Sie beispielsweise einer Variablen zuweisen.

```
$lambda = function($a) { return $a / 2; };
```

Und dann können Sie die Funktion folgendermaßen aufrufen:

```
$ergebnis = $lambda(5);
```

Achtung

Wichtig ist der Strichpunkt, um die Zuweisung zu `$lambda` abzuschließen. Ansonsten erhalten Sie eine Fehlermeldung.

Das ist praktisch, um Funktionen als Parameter zu übergeben. Beispielsweise bei der von PHP vordefinierte Funktion `array_map()`. Über `array_map()` können Sie bestimmte Verarbeitungsschritte *für alle Elemente eines Arrays* durchführen. Als ersten Parameter erwartet `array_map()` eine Funktion, die bestimmt, was mit den einzelnen Elementen des Arrays geschehen soll, als zweiten Parameter das Array.

Und wenn diese Funktion, die man an `array_map()` übergibt, ansonsten nicht benötigt wird, empfiehlt sich die Verwendung einer anonymen Funktion:

Listing 5.26: Lambda-Funktion (lambda.php)

```
$lambda = function($a) { return $a / 2; };
$zahlen = array (4, 33, 2);
$ergebnis = array_map($lambda, $zahlen);
print_r($ergebnis);
```

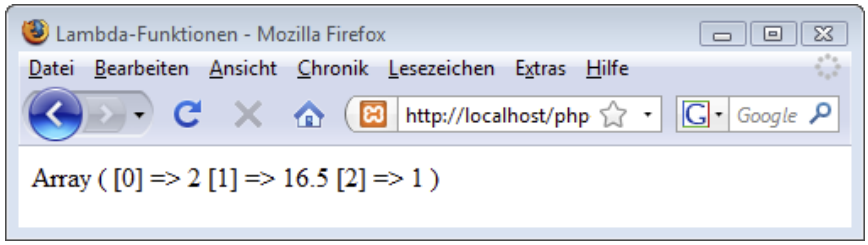


Abbildung 5.17: Alle Elemente des Arrays werden durch 2 geteilt.

Dies ließe sich auch verkürzt schreiben, indem Sie die anonyme Funktion direkt bei `array_map()` angeben:

Listing 5.27: Noch kompakter – die anonyme Funktion wird direkt array_map() übergeben.

```
$zahlen = array (4, 33, 2);
$ergebnis = array_map(function($a) { return $a / 2; }, $zahlen);
print_r($ergebnis);
```

Hinweis

Vor PHP 5.3 hatten Sie hier nur die Möglichkeit, eine Funktion über `create_function()` zu erstellen.

Wenn man Lambda-Funktionen an externe, außerhalb der Funktion existierende Variablen bindet, spricht man von *Closures*. Die Variablen, die benutzt werden sollen,

werden mit `use` angegeben. Das Besondere dabei ist, dass die Variable an die Closure gebunden wird, wenn diese definiert wird. Wird die Variable dann geändert, hat das keinerlei Einfluss:

Listing 5.28: Closure (closures.php)

```
01 $faktor = 2;
02 $lambda = function($a) use($faktor) {
03   return $a / $faktor;
04 };
05 $erg1 = $lambda(33);
06 echo "\$erg1 ist $erg1<br />\n";
07
08 $faktor = 3;
09 $erg2 = $lambda(33);
10 echo "\$erg2 ist $erg2<br />\n";
```

Im Beispiel wird in Zeile 2 eine Closure definiert und die Variable `$faktor`, die in Zeile 2 auf 2 gesetzt ist, mit `use` verwendet. Wie zu erwarten ist `$erg1` damit 16,5.

In Zeile 8 wird der Faktor geändert, nämlich `3` gesetzt. Das hat jedoch auf die Closure keine Auswirkung, auch `$erg2` ist 16,5.

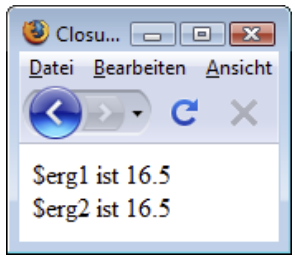


Abbildung 5.18: Die Änderung des Inhalts von $faktor nach Definition der Closure hat keinen Einfluss.

Lambda-Funktionen und Closures sind Features, die Sie sicher nicht ständig brauchen werden. Praktisch sind Lambda-Funktionen aber beispielsweise für Sortierfunktionen und bei Funktionen, die Arrayinhalte ändern und so sollten Sie wissen, worum es sich dabei handelt.

5.4 Klassen und Objekte

Sie haben gerade gesehen, wie Sie über Funktionen Ihre PHP-Skripte besser organisieren können. Die Objekte und Klassen, um die es in diesem Abschnitt geht, gehen noch einen Schritt weiter: Sie erlauben es Funktionen, die hier aber Methoden genannt werden, und Variablen, die hier Eigenschaften genannt werden, zu bündeln

und gemeinsam wiederzuverwenden. Das Ziel: gut organisierter, leicht wieder verwendbarer und gut wartbarer Code. Und damit kommen wir zur Objektorientierung.

5.4.1 Objektorientierte Programmierung

Was Sie bisher kennen gelernt haben, war die prozedurale Programmierung, daneben kann man in PHP auch objektorientiert programmieren. Objektorientierte Programmierung ist eine fortgeschrittene Programmiertechnik. Warum aber als Einsteiger sich schon mit Objektorientierung auseinandersetzen?

Prinzipiell werden Sie am objektorientierten Ansatz nicht vorbeikommen, da einige grundlegende Funktionen von PHP objektorientiert konstruiert sind. Also müssen Sie Klassen und Objekte verwenden und sollten deswegen die etwas andere Syntax verstehen.

Größere PHP-Anwendungen wie Content Management-System o. Ä. sind häufig objektorientiert programmiert. Wenn Sie Erweiterungen oder Modifikationen hierfür schreiben möchten, müssen Sie diese dann ebenfalls objektorientiert programmieren. Das Gleiche gilt auch für PHP-Frameworks.

Nun zu den Prinzipien der Objektorientierung. In der objektorientierten Herangehensweise ist erst einmal alles ein Objekt. Wenn Sie einen Shop programmieren, so können Sie beispielsweise ein Objekt *Warenkorb* haben, ein anderes Objekt *Kunde* und wieder ein anderes Objekt *Produkt*. Bei einem anderen Projekt könnten Sie es mit einem Objekt zu tun haben, das die Aufgabe eines Dateimanagers erfüllt, oder Sie könnten Objekte haben, über die Sie die Kommunikation mit der Datenbank regeln.

Diese Objekte haben dann verschiedene *Eigenschaften* (Variablen) und *Methoden* (Funktionen). Eigenschaften und Methoden gehören zu einem Objekt und machen das Objekt als Ganzes aus. Das Objekt *Warenkorb* hat beispielsweise als Eigenschaften die jeweilige Nummer des Kunden und eine Liste von Artikeln. Außerdem hat es sicher Methoden, um den Inhalt des Warenkorbs anzeigen zu lassen, einen Artikel zum Warenkorb hinzuzufügen oder zu löschen.

Die einzelnen konkreten Warenkörbe der Kunden werden aber unterschiedlich sein: Erst einmal unterscheiden sie sich natürlich in der Kundennummer, dann aber auch im Inhalt des Warenkorbs etc. Gemeinsam ist ihnen aber, dass sie eine Kundennummer und einen Inhalt mit Artikel(n) haben. Es gibt also einerseits das allgemeine Schema – das, was alle Warenkörbe gemeinsam haben – und den konkreten Fall – das, was ein individueller Warenkorb beinhaltet.

Diese Unterscheidung wird auch in der objektorientierten Programmierung abgebildet: Das, was alle Warenkörbe gemeinsam haben, also das Konzept dahinter, wird durch die Definition einer *Klasse* festgelegt. Der Einzelfall – der individuelle Warenkorb – ist hingegen dann eine Instanziierung der Klasse und konkret ein *Objekt*.

Das ist vielleicht auch der größte Unterschied zur prozeduralen Programmierung: Prozuderal legen Sie sofort los, beim objektorientierten Ansatz erstellen Sie zuerst eine Klassendefinition als Schablone. Dann erstellen Sie ein konkretes Objekt mit den spezifischen Daten.

5.4.2 Methoden und Eigenschaften

Ein erstes Beispiel zeigt die Definition einer Klasse. Eine Klasse erstellen Sie mit dem Schlüsselwort `class` gefolgt vom Namen der Klasse. Üblicherweise sollten Klassennamen mit einem Großbuchstaben beginnen.

```
class Kunde {}
```

Fehlen natürlich noch Eigenschaften und Methoden:

```
class Kunde
{
  public $name;
  public function halloSagen()
  {
    echo "Hallo";
  }
}
```

Innerhalb der Klasse wird eine Eigenschaft `$name` definiert. Außerdem gibt es eine Methode `halloSagen()`. Für Methoden verwenden Sie das bereits bekannte Schlüsselwort `function` gefolgt vom Namen der Methode und runden Klammern. In geschweiften Klammern steht der Rumpf der Methode.

Tipp

Außerdem steht vor allen Eigenschaften und Methoden das Schlüsselwort `public`. Es regelt den Zugriff auf die entsprechende Eigenschaft/Methode (mehr hierzu in Kapitel 9).

Wenn Sie das obige Skript ausführen, passiert noch nichts, weil Sie zwar die Klasse definiert, aber noch kein konkretes Objekt erzeugt haben.

Ein konkretes Objekt erstellen Sie mit dem Schlüsselwort `new` und dem Namen der Klasse. Dies weisen Sie einer Variablen zu, über die Sie das Objekt der Klasse dann ansprechen können.

```
$neuerKunde = new Kunde();
```

Dann können Sie die einzelnen Eigenschaften setzen:

```
$neuerKunde->name = "Anja";
```

Schreiben Sie dafür den Namen der Variablen, die den Verweis auf das Objekt speichert, den Pfeiloperator gefolgt von der Eigenschaft *ohne Dollarzeichen*.

Genauso greifen Sie auch auf die Methoden der Klasse zu, müssen hier aber hinter dem Methodennamen die runden Klammern notieren:

```
$neuerKunde->halloSagen();
```

Die Methoden, die Sie in Klassen definieren, können – genauso wie Funktionen – selbstverständlich auch Parameter übernehmen.

Hier sehen Sie das ganze Skript noch einmal im Gesamtzusammenhang:

Listing 5.29: Ein Objekt der Klasse Kunde wird erstellt (kunde_beispiel.php).

```
01 class Kunde
02 {
03   public $name;
04   public function halloSagen()
05   {
06     echo "Hallo";
07   }
08 }
09 $neuerKunde = new Kunde();
10 $neuerKunde->name = "Anja";
11 $neuerKunde->halloSagen();
12 echo " ";
13 echo $neuerKunde->name;
```

Das Skript gibt `Hallo Anja` aus.

Das soll als erster kurzer Hinweis auf die objektorientierte Programmierung mit PHP genügen, Sie werden dazu im Folgenden weitere Beispiele finden und Kapitel 9 widmet sich dann ausschließlich diesem Thema und stellt auch fortgeschrittene Möglichkeiten vor.

5.5 Unterstützung bei der Fehlersuche

Wenn Sie selbst Skripte ausprobiert, vielleicht auch an ein paar Stellen angepasst haben, dann kennen Sie eines inzwischen wahrscheinlich gut: die Fehlermeldungen. Um Fehler besser zu finden oder gleich zu vermeiden, helfen erst einmal zweierlei: ein übersichtlich gestalteter Code und der richtige Editor. Außerdem erhalten Sie Tipps, wie Sie die Fehlermeldungen besser verstehen.

5.5.1 Leerzeichen und Einrückungen

Inzwischen sind die Skripte schon komplexer geworden und Sie haben viele wichtige Sprachkonstruktionen kennen gelernt. Der richtige Zeitpunkt, sich noch einmal anzusehen, wie es sich mit Leerzeichen und Einrückungen in den PHP-Skripten verhält.

Das folgende Beispiel zeigt – anhand des Codes des kleinen Einmaleins (Listing 5.13) –, wo Sie überall beliebig viele Leerzeichen setzen können, ohne dass PHP bei der Ausführung des Skripts stört.

Listing 5.30: An vielen Stellen sind beliebig viele Leerzeichen möglich (einmaleins_luftig.php).

```
for   (   $i   =   1   ;    $i   <=   10   ;   $i++   )   {
  echo   "<tr>\n"   ;
  for    (   $j   =   1   ;   $j   <=   10   ;   $j++   )   {
    $zahl   =   $i   *   $j   ;
    echo   "\t<td>$zahl</td>\n"   ;
  }
  echo    "</tr>\n"   ;
}
```

Außerdem könnte überall, wo ein Leerzeichen steht, auch ein Zeilenumbruch stehen.

An all diesen Stellen können Leerzeichen stehen, Sie können sie aber auch weglassen und ebenfalls die Zeilenumbrüche entfernen, wie das folgende Listing zeigt.

Listing 5.31: Ganz ohne Zeilenumbrüche (einmaleins_eine_zeile.php)

```
for($i=1;$i<=10;$i++){echo"<tr>\n";for($j=1;$j<=10;$j++)
{$zahl=$i*$j;echo"\t<td>$zahl</td>\n";}echo"</tr>\n";}
```

Prinzipiell haben Sie hier also relativ viel Freiheit. Es gibt jedoch Ausdrücke, die zusammenstehen müssen. Nicht durch Leerzeichen trennen dürfen Sie beispielsweise den Inkrementoperator (`$i++`), das Newline-Zeichen `\n` oder Operatoren wie `&&`, `<=` oder `==` etc.

Sie sehen: Für die Ausführung der Skripten sind die meisten Leerzeichen und Zeilenumbrüche nicht relevant. Sie sind aber ganz wichtig, damit der Code lesbar ist, und helfen bei der Fehlersuche. Wichtig ist insbesondere, dass man bei Anweisungsblöcken erkennt, wozu sie gehören. Das geschieht über die korrekten Einrückungen.

Hinweis

Im Buch wird aus Platzgründen der Code immer nur um zwei Zeichen eingerückt, Sie sollten der besseren Lesbarkeit Ihrer Skripten zuliebe aber besser die doppelte Anzahl nehmen, also vier Leerzeichen.

Für die Anordnung der geschweiften Klammern gibt es mehrere Möglichkeiten. Die öffnende Klammer kann in eine neue Zeile geschrieben werden oder aber sie kann in dieselbe Zeile geschrieben werden: Hier im Buch folge ich diesbezüglich den PEAR-Coding-Standards, bei Funktionen schreibe ich die öffnende Klammer in eine neue Zeile, bei Kontrollstrukturen hingegen in die Zeile, in der die Kontrollstruktur beginnt:

```
function beispiel()
{
  if () {
  } else {
  }
}
```

Wichtig ist, dass die schließenden Klammern immer so angeordnet sind, dass der Bezug ganz eindeutig ist.

Hinweis

Es gibt verschiedene Best-Practice-Anweisungen, wie übersichtlicher PHP-Code aussehen sollte. Eine gute Anlaufstelle hierfür sind zum einen die gerade erwähnten PEAR Coding Standards unter *http://pear.php.net/manual/de/standards.php* und die Webseite *http://php-coding-standard.de/*, die sich speziell diesem Thema widmet.

Den Code gut einzurücken und übersichtlich zu gestalten, erleichtert die Fehlersuche bei Problemen, eine weitere Hilfe bietet Ihnen der richtige Editor.

5.5.2 Editor mit mehr Fähigkeiten

Ebenfalls sehr empfehlenswert ist es, einen Editor einzusetzen, der etwas mehr kann, als der einfache Texteditor: Zumindest sollte er Syntaxhervorhebung bieten, zusammenhängende Klammern markieren können und die Zeilennummer anzeigen – das ist die Minimalanforderung.

Ein kostenloser Editor, der diese Funktionen bereitstellt, ist Scite. Es gibt ihn für Windows und Linux. Sie finden ihn unter *http://www.scintilla.org/SciTE.html*.

Tipp

Um eine PHP-Datei in Scite zu öffnen, müssen Sie beim Öffnen rechts neben dem Dateinamensfeld anstelle von ALL SOURCE die Option ALL FILES wählen. Dann werden Ihnen die PHP-Dateien angezeigt und Sie können die gewünschte auswählen.

Hinweis

Auch für Mac OS X gibt es viele nützliche Editoren, beispielsweise aptana (*http://www.aptana.com/*). Weitere Tipps liefert die Software-Suche bei Heise unter *http://www.heise.de/software/*.

Scite hebt die einzelnen Bestandteile Ihres Codes in unterschiedlichen Farben hervor.

Wenn die Farbzuweisungen an einer Stelle nicht mehr stimmen, ist das meist ein Hinweis, dass Sie einen Fehler in Ihrem Code haben. Vergessen Sie beispielsweise das schließende Anführungszeichen bei einem String, sind alle weiteren Zeilen bis zum nächsten Anführungszeichen grün eingefärbt wie Strings und folgen nicht dem sonstigen Farbschema, dass Klammern beispielsweise blau sind, Schlüsselwörter wie `else` lila etc.

Außerdem können Sie sehen, welche Klammern zusammengehören. Wenn Sie eine geschweifte Klammer mit der Maus markieren, wird die zugehörige Klammer blau hervorgehoben.

Codezeilen lassen sich ausblenden – durch Klick auf das Minus vor einem Codeblock. Dann wird aus dem Minus ein Plus, das sich durch einen erneuten Klick wieder expandieren lässt.

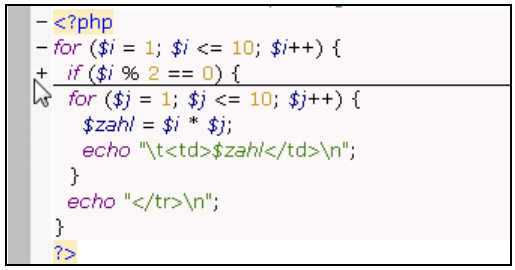


Abbildung 5.19: Codeblöcke können durch einen Klick auf das Minus eingeklappt werden.

Bei Fehlermeldungen erhalten Sie in PHP die Nummer der Zeile ausgegeben, in der PHP auf den Fehler gestoßen ist. Um diese zuzuordnen, können Sie sich in Scite die Zeilennummer anzeigen lassen über VIEW/LINE NUMBERS.

5.6 Fehlersuche – der Parse Error

Ein übersichtlich gestalteter Code und der richtige Editor helfen bei der Fehlersuche. Wichtige Hinweise auf den Fehler liefern die Fehlermeldungen – wenn man weiß, wie man sie zu interpretieren hat. Häufig begegnet ist Ihnen sicher schon der Parse

Error, ein Fehler, der beim Parsen des Skripts auftritt. Beim Parse Error wird das Skript gar nicht verarbeitet. Wenn Sie den zugrunde liegenden (X)HTML-Quellcode ansehen, sehen Sie, dass nicht einmal das (X)HTML-Grundgerüst ausgegeben wird. Es erscheint eine Fehlermeldung und mehr nicht.

Abbildung 5.20: Parse Error

5.6.1 Fehlendes Anführungszeichen

Nehmen wir als Beispiel einmal das Listing *einmaleins_farbig.php* (Listing 5.14). Angenommen, Sie haben ein Anführungszeichen vergessen – ein typischer Fall für einen Parse Error:

```
15   if ($i % 2 == 0) {
16     echo "<tr class='gerade'>\n;
17   } else {
18     echo "<tr>\n";
19   }
```

Sie sehen einen Ausschnitt aus dem fehlerhaften Listing. Die Nummerierung beginnt hier bei 14, was der Nummerierung der Zeilen im Skript entspricht. Am Ende der Zeile 16 fehlt ein Anführungszeichen. Sie erhalten dann die oben in Abbildung 5.20 gezeigte Fehlermeldung: PHP meldet ein unerwartetes > in Zeile 18. Das bedeutet, PHP meldet einen Fehler zwei Zeilen später. Warum das?

Der String wird für PHP beim nächsten Auftreten eines Anführungszeichens geschlossen. Der String umfasst also:

```
16     echo "<tr class='gerade'>\n;
17   } else {
18     echo "<tr>\n";
```

Das heißt, der String geht bis `echo "`. Dann folgt ein Kleiner-als-Zeichen und `tr`, das PHP als Konstante interpretiert. Erst beim schließenden >, das PHP als größer als liest, passt es nicht mehr und an dieser Stelle meldet PHP den Fehler.

Rechnen Sie also bei einem Parse Error damit, dass Sie den Fehler früher gemacht haben. Fehler mit fehlenden Anführungszeichen entdeckten Sie gut über die Funktion zum Syntaxhervorheben Ihres Editors.

Hilfreich zur Vermeidung solcher Fehler ist es, sich bei Strings, die man schreibt, direkt beide Anführungszeichen zu notieren – am besten gleich mit dem abschließenden Anführungszeichen und danach den Inhalt zu ergänzen.

Dabei müssen Sie dann nur noch darauf achten, dass Sie, wenn Sie für (X)HTML-Attributwerte weitere Anführungszeichen brauchen, die Anführungszeichen maskieren oder gleich einfache nehmen.

5.6.2 Vergessene geschweifte Klammern

Vergessene geschweifte Klammern sind ebenfalls ein beliebter Fehler – und auch hier stimmt oft die Nummer, die der PHP-Interpreter bei der Fehlermeldung angibt, nicht mit der Zeile überein, in der Sie eigentlich die Klammer vergessen haben.

Ein Beispiel zeigt dies:

```
15   if ($i % 2 == 0) {
16     echo "<tr class='gerade'>\n";
17   } else {
18     echo "<tr>\n";
19
20   for ($j = 1; $j <= 10; $j++) {
21     $zahl = $i * $j;
22     echo "\t<td>$zahl</td>\n";
23   }
24   echo "</tr>\n";
25 }
26 ?>
27 </table>
28 </body>
29 </html>
```

Im Beispiel fehlt in Zeile 19 die schließende Klammer des `else`-Zweigs. Der PHP-Interpreter ordnet die Klammern neu zu. Als schließende Klammer für `else` wird die schließende Klammer in Zeile 25 ausgemacht. Damit fehlt aber noch die schließende Klammer der umfassenden `for`-Schleife. Sie erhalten die in Abbildung 5.21 gezeigte Fehlermeldung.

Abbildung 5.21: Eine Fehlermeldung bei einer fehlenden geschweiften Klammer

Die Fehlermeldung besagt, dass das Ende des Skripts unerwartet kommt. Und angezeigt wird der Fehler für die letzte Zeile des Skripts. Im Beispiel ist das die letzte Zeile mit dem schließenden `</html>`. Es könnte aber noch weiter hinten sein – wenn weitere leere Zeilen am Ende anschließen. Kontrollieren Sie in diesem Fall Ihre Klammern.

Aber natürlich gibt es auch einfachere Fälle: Fälle, in denen die Zeile, in der der Fehler vom PHP-Interpreter gemeldet wird, auch wirklich mit der Zeile übereinstimmt, in der der Fehler aufgetreten ist. Hierzu folgendes Beispiel:

```
15   if ($i % 2 == 0) {
16     echo "<tr class='gerade'>\n";
17   else {
18     echo "<tr>\n";
19   }
20   for ($j = 1; $j <= 10; $j++) {
```

Hier fehlt in Zeile 17 die schließende Klammer vom `if`-Zweig, bevor das `else` losgeht.

Abbildung 5.22: Dieses Mal ist der Fehler wirklich in der Zeile 17.

Genau das meldet der PHP-Interpreter auch: »unexpected T_ELSE« in Zeile 17. Erwartet wird an dieser Stelle nicht `else`, sondern es müsste eine Klammer stehen. Und warum meldet der PHP-Interpreter `T_ELSE` und nicht `else`? Intern werden bestimmte Bestandteile von PHP als Token bezeichnet. Token heißt so viel wie »Zeichen/Marke«. In den meisten Fällen ist es einfach: Sie streichen das `T_` am Anfang und wissen, worum es sich handelt.

Tipp

Eine Auflistung der Token finden Sie unter *http://www.php.net/manual/en/tokens.php*.

Tipp

Und noch zwei Tipps zur Fehlersuche: Im Index dieses Buchs finden Sie alle Fehlermeldungen, die besprochen wurden, unter dem Eintrag »Fehlermeldung« aufgeführt. Wenn Ihnen ansonsten der Text einer Fehlermeldung sehr kryptisch vorkommt, hilft oft eine Internetsuche. Bei dieser geben Sie den ersten Teil Ihrer Fehlermeldung wörtlich ins Suchfeld ein – selbstverständlich aber ohne Angabe der Datei und der Zeilennummer, weil das von Skript zu Skript variiert.

5.6.3 Mehr Fehlertypen

Neben dem Parse Error gibt es folgende Fehlertypen:

- FATAL ERROR: Der fatale Fehler deutet auf ein ernsthaftes Problem hin, wenn man z. B. eine Funktion verwendet, die es nicht gibt. In diesem Fall wird das Skript ausgeführt bis zur Stelle mit dem fatalen Fehler, und dann die weitere Verarbeitung abgebrochen.
- WARNING: Die Warnung ist ein Hinweis, dass etwas mit Ihrem Programm nicht so ist, wie es sein sollte. Sie erhalten etwa eine Warnung, wenn Sie versuchen, `foreach` bei einem String zu benutzen. Bei Warnungen wird der Code ansonsten normal ausgeführt.
- NOTICE: Diese erhalten Sie beispielsweise, wenn Sie eine nicht definierte Variable benutzen. Die entsprechende Konstante heißt `E_NOTICE`.
- `Strict Standards` beinhalten Ermahnungen zu Ihrem Programmierstil. Die Konstante, die die Ausgabe dieser Meldungen steuert, heißt `E_STRICT`.
- `DEPRECATED` ist neu in PHP 5.3. Diese Meldung erhalten Sie, wenn Sie Features verwenden, die eigentlich nicht mehr erwünscht sind und die es in PHP 6 nicht mehr geben wird.

Das Skript wird nur beim Fatal Error abgebrochen, ansonsten wird es trotz Meldung ausgeführt.

Das sind die Fehler, die Ihnen PHP anzeigen *kann*. Welche aber wirklich angezeigt werden, ist eine Konfigurationssache. Sie können über Einstellungen zum einen bestimmen, welche Fehler ausgegeben werden sollen, und zum anderen, wo sie ausgegeben werden sollen.

`error_reporting` steuert, welche Fehler angezeigt werden sollen. Dies wird über Konstanten geregelt. Hier ein paar gebräuchliche Varianten:

- `E_ALL` zeigt alle Fehler an – bis auf die `E_STRICT`-Fehler.
- `E_ALL | E_STRICT` zeigt alle Fehler und die vom Strict-Standard ebenfalls an.
- `E_ALL & ~E_NOTICE` zeigt alle Fehler bis auf die Hinweise und die vom Strict-Standard an.

`display_errors` bestimmt, ob die Fehler im Browser angezeigt werden sollen.

- `display_errors = On` ist die richtige Wahl für ein System, auf dem man entwickelt.
- `display_errors = Off` die richtige Wahl für einen Produktivbetrieb.

`log_errors` legt fest, ob die Fehler in eine Logdatei (Standard-Errorlog des Webservers) geschrieben werden sollen.

- `log_errors = off` ist geeignet für Entwicklungssysteme, wo Sie sich die Fehler direkt anzeigen lassen.
- `log_errors = on` ist geeignet für Produktivsysteme.

Welche Fehler angezeigt werden sollen, können Sie auch direkt in einzelnen Skripten regeln. Über folgende Zeile schalten Sie die Anzeige von Fehlern aus:

```
error_reporting(0);
```

Und über die folgende Zeile in einem Skript sorgen Sie dafür, dass alle Fehler außer `E_NOTICE` und `E_STRICT` angezeigt werden.

```
error_reporting(E_ALL ^ E_NOTICE);
```

So sehr Fehlermeldungen erst einmal ärgerlich sind, so sind sie doch äußerst nützlich. Wesentlich mühsamer ist es, Fehler zu finden, wenn Sie keine Fehlermeldung erhalten.

Im nächsten Kapitel geht es jetzt um die Vorstellung von weiteren nützlichen Funktionen zur Arbeit mit Strings oder Arrays und zur Ausgabe von Datums/Zeitinformationen.

6 Funktionen für Strings, Arrays, Datum und mehr

In den bisherigen Kapiteln haben Sie schon einzelne Beispiele für nützliche Funktionen gesehen, die Ihnen zur Verfügung stehen. Dieses Kapitel widmet sich jetzt den in PHP vordefinierten Funktionen im Detail. Vorgestellt wird Nützliches für die Arbeit mit Strings, mit Arrays und Datumsfunktionen. Zuerst erfahren Sie aber, wo Sie weitere Informationen zu Funktionen nachschlagen können.

6.1 Funktionen im PHP-Manual

Das PHP-Manual ist die offizielle Dokumentation zu PHP. Hier finden Sie auch Informationen zu (fast) allen Funktionen. Kennen Sie den Namen einer Funktion, können Sie ganz rasch über die offizielle PHP-Seite zur entsprechenden Seite im PHP-Manual gelangen. Geben Sie in das Suchfeld bei SEARCH FOR den Namen der Funktion ein, nach der Sie suchen, und Sie gelangen zur Dokumentation.

Abbildung 6.1: Übersichtlich: die Funktionsreferenz im Online-Manual

Die Informationen zu den einzelnen Funktionen haben einen vordefinierten Aufbau:

- Zuerst steht der Name der Funktion.
- In Klammern darunter sehen Sie, in welchen PHP-Versionen die Funktion zur Verfügung steht.
- Unter BESCHREIBUNG erhalten Sie Informationen dazu, welche Parameter die Funktion erwartet und was der Rückgabewert ist.

Bei der Funktion `date()` sieht diese Beschreibung beispielsweise folgendermaßen aus:

```
string date  ( string $Format  [, int $Timestamp  ] )
```

Vor dem Namen der Funktion ist der Rückgabewert angegeben. `date()` gibt einen String zurück. Falls eine Funktion nichts zurückgibt, steht an dieser Stelle `void`.

In runden Klammern stehen die Parameter, die die Funktion erwartet. `date()` erwartet einen Parameter, der ebenfalls ein String ist (genauere Informationen zu den Parametern folgen im Manual dann weiter unten). In eckigen Klammern stehen weitere mögliche Parameter, die jedoch optional sind, das heißt, dass Sie sie nicht einzusetzen brauchen.

Neben den gängigen Variablentypen (siehe Kapitel 4), die hier verwendet werden, gibt es noch drei weitere, die keine echten Variablentypen sind, sondern nur in der Dokumentation auftauchen, um die Beschreibungen zu vereinfachen:

- `mixed` zeigt an, dass eine Funktion unterschiedliche Variablentypen erwartet, das können alle sein oder nur ein Teil der in PHP vorhandenen. `mixed` ist beispielsweise bei `var_dump()` angegeben, das man ja bei beliebigen Datentypen einsetzen kann.
- `number` steht für Float oder Integer.
- `callback` steht für benutzerdefinierte Callback-Funktionen. Das sind Funktionen, die anderen Funktionen als Parameter übergeben werden. So erwartet beispielsweise `array_map()` als ersten Parameter eine Funktion. Hier können dann anonyme Funktionen (siehe Kapitel 5) eingesetzt werden.

Nach der Beschreibung finden Sie weitere Erläuterungen zur Verwendung und Beispiele. Sehr nützlich sind auch die Kommentare von Benutzern, die am Schluss folgen, die jedoch von unterschiedlicher Qualität sind.

Tipp

Finden Sie eine gewünschte Information nicht, sollten Sie einmal auf die englische Version umschalten, die mitunter vollständigere/aktuellere Informationen enthält.

6.2 Funktionen für Strings

Zur Bearbeitung von Strings gibt es viele nützliche vordefinierte Funktionen.

Um einzelne Zeichen aus einem String zu ermitteln oder auch zu ersetzen, brauchen Sie keine Funktionen, das können Sie direkt machen, indem Sie den String so behandeln, als wäre er ein Array. Auf ein einzelnes Zeichen eines Strings greifen Sie zu, indem Sie den Namen der Variable notieren und dahinter in eckigen Klammern die Position des Zeichens. Die Zählung beginnt dabei wie bei Arrays mit 0.

Listing 6.1: Einzelne Zeichen eines Strings ausgeben und ändern (einzelne_zeichen.php)

```
$tier = "Maus";
$erster = $tier[0];
echo "$tier beginnt mit $erster<br />\n";
$tier[0] = "L";
echo $tier;
if (!isset($tier[4])) {
  echo "<br />\n $tier hat weniger als 5 Buchstaben";
}
```

Im Beispiel wird zuerst das erste Zeichen ausgelesen und ausgegeben. Dann wird es ersetzt und das Ergebnis wieder ausgegeben. Sie können diese Methode des Zugriffs auf einzelne Zeichen aus einem String über eckige Klammern auch einsetzen, um zu überprüfen, wie lang ein String ist. `isset()` prüft, ob eine Variable gesetzt, d.h. definiert ist, und gibt `true` oder `false` zurück. Wenn `$tier[4]` nicht gesetzt ist, bedeutet es, dass der String weniger als 5 Buchstaben hat, da die Zählung wie gesagt bei 0 beginnt.

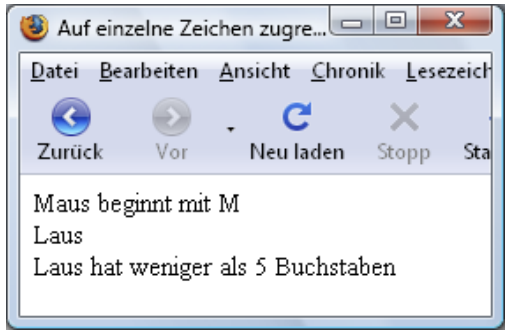


Abbildung 6.2: Auf einzelne Buchstaben eines Strings kann man ohne Funktion direkt mit eckigen Klammern zugreifen.

Mit den eckigen Klammern können Sie nur feststellen, ob an einer Position noch ein Zeichen vorhanden ist oder nicht. Um hingegen die Länge eines Strings zu ermitteln, brauchen Sie die Funktion `strlen()`.

Listing 6.2: Die Länge eines Strings ermittelt `strlen()` (strlen.php).

```
$name = " Müller-Lüdenscheidt ";
$laenge = strlen($name);
echo "$name ist $laenge Zeichen lang (inkl. Leerzeichen)<br />\n";
```

Im Beispiel wird die Länge eines Strings ermittelt und ausgegeben. Der String im Beispiel beinhaltet Leerzeichen, die mitgezählt werden. Deswegen ist das Ergebnis 21 Zeichen. Zur Entfernung von Leerzeichen können Sie `trim()`, `rtrim()` und `ltrim()` benutzen.

`trim()` entfernt Leerzeichen am Anfang und Ende eines Strings. Als Leerzeichen zählen dabei auch das Newline-Zeichen oder der Tabulator. `trim()` ist praktisch, um mögliche *Leerzeichen von Eingaben* zu entfernen. Sie wollen schließlich »Müller« in Ihrer Datenbank (oder sonst wo) speichern und nicht » Müller « etc. Schließlich werden Sie auch beim Auslesen prüfen, ob der Name »Müller« ist und nicht » Müller «.

Listing 6.3: Leerzeichen entfernen mit trim() (trim.php)

```
01 $nn = "   Müller     ";
02 $vn = "\tBerenice\n ";
03 echo "<pre>";
04 echo "Vor trim:
05          Name: $nn,
06          Vorname $vn.";
07 $nn = trim($nn);
08 $vn = trim($vn);
09 echo "Nach trim:
10          Name: $nn.
11          Vorname: $vn.";
12 echo "</pre>";
```

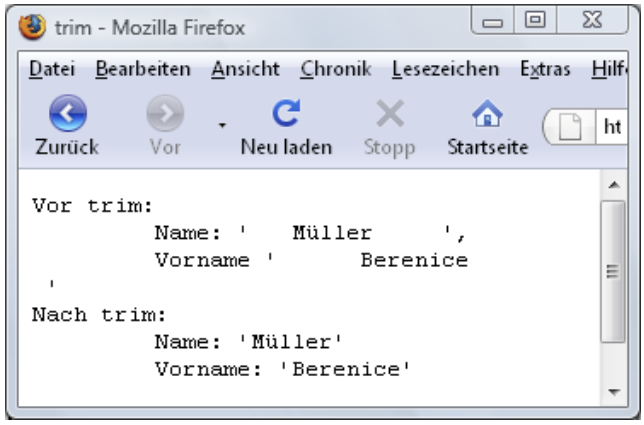


Abbildung 6.3: trim() im Einsatz

Im Beispiel wird das (X)HTML-Element `pre` eingesetzt, damit die Leerzeichen auch im Browser sichtbar sind.

Sie können `trim()` auch benutzen, um andere Zeichen als Leerzeichen zu entfernen. Hierfür geben Sie als zweiten Parameter eine Liste der Zeichen an, die Sie löschen möchten. Im folgenden Beispiel sollen das Hochkomma und der Punkt am Anfang und Ende der Zeichenkette entfernt werden.

```
$str = "'O'Brien'...";
echo trim($str, "'.");
```

Das Ergebnis ist *O'Brien*. Das ' im String bleibt unberührt. Um alle Vorkommen von bestimmten Zeichen zu entfernen, gibt es `str_replace()` (Abschnitt 6.2.2).

Zum Entfernen von Zeichen nur vorne oder nur hinten existieren zwei weitere Funktionen: `rtrim()` entfernt die gewünschten Zeichen rechts (hinten) und `ltrim()` links, d.h. vorne.

Wo wir gerade bei Leerzeichen sind – die Funktion `empty()` macht mehr, als man dem Namen nach vermuten würde. Sie liefert wahr zurück, wenn ein String leer ist. Aber ebenso liefert `empty()` wahr, wenn eine nicht gesetzte Variable überprüft wird oder eine Variable, die den Wert 0 hat. Im Allgemeinen gibt `empty()` wahr zurück, wenn der Wert `false` ergibt.

Listing 6.4: Überprüfung mit empty() (empty.php)

```
01 $str    = "   ";
02 $str    = trim($str);
03 $z      = 0;
04 $falsch = false;
05 if (empty($str)) {
06   echo "das ist zu mager - ";
07 }
08 if (empty($gibtsnicht)) {
09   echo " auch nicht besser - ";
10 }
11 if (empty($z)) {
12   echo " von nichts kommt nichts - ";
13 }
14 if (empty($falsch)) {
15   echo " wieder nix ";
16 }
```

Alle vier Überprüfungen klappen, es werden also alle Texte ausgegeben.

6.2.1 Mehr Optionen für die Ausgabe

In diesem Abschnitt geht es um die formatierte Ausgabe über `printf()` und die Formatierung von Zahlen.

Formatierte Ausgabe über printf()

Mit `print` oder `echo` geben Sie die Dinge so aus, wie sie sind. Eine *formatierte Ausgabe* ist hingegen mit `printf()` möglich. `printf()` erwartet als ersten Parameter einen Formatierungsstring, darauf können mehrere Parameter verschiedenen Typs folgen.

Die Formatierungsstrings sind immer durch das Prozentzeichen (%) gekennzeichnet, deswegen können Sie den Formatierungsstring mit »normalem« Text kombinieren. `%s` steht dabei für einen String und `%d` für einen Integer.

```
printf("%s ist %d Jahre alt", "Ben", 4);
```

gibt aus »Ben ist 4 Jahre alt«.

Wenn Sie einen anderen Typ übergeben, als Sie im Formatierungsstring angegeben haben, führt `printf()` die Konvertierung durch:

```
printf("%d", 2.567);
```

Hier steht als Formatierungsstring `%d`, d.h. eine ganze Zahl, übergeben wird allerdings eine Fließkommazahl. Ausgegeben wird 2 – die Stellen hinter dem Komma werden abgeschnitten.

Außerdem können Sie angeben, wie lange der zurückgegebene String mindestens sein soll und was als Füllzeichen verwendet werden soll:

```
printf("%03d", 7);
```

Hier wird durch die `03` bestimmt, dass der String mindestens 3 Zeichen lang sein und als Füllzeichen `0` benutzt werden soll. Ausgegeben wird in diesem Fall »007«.

Tipp

Wenn Sie kein Füllzeichen angeben, werden Leerzeichen benutzt. Andere Zeichen als 0 müssen Sie über ein `'` kennzeichnen.

`printf()` kann zur Formatierung von Fließkommazahlen benutzt werden: Für Floats wird `%f` benutzt und Sie können angeben, wie viel Dezimalstellen angezeigt werden. Diese geben Sie hinter einem Punkt an:

```
printf("%.2f", 34.567); //34.57
```

Durch `.2f` beschränken Sie die Ausgabe auf zwei Dezimalzeichen.

Tipp

Sie können natürlich gleichzeitig die Gesamtlänge des Strings und die Anzahl der Nachkommastellen bestimmen.

```
printf("%07.2f", 34.567);
```

Dabei bestimmt `7` nicht – wie häufig fälschlich angenommen wird – die Anzahl der Zeichen vor dem Punkt, sondern *die Gesamtlänge des Strings*. Ausgegeben wird hier 0034.57 – der Punkt zählt mit.

Praktisch an `printf()` ist, dass Sie die einzelnen Bestandteile – den Formatierungsstring und die zu formatierenden Inhalte – über Variablen bestimmen können. So kann die Art der Ausgabe abhängig von Bedingungen gemacht werden.

Listing 6.5: Je nach Bedingung unterschiedlich formatierte Ausgabe (printf.php)

```
$genau    = true;
$ergebnis = 9.123456;
if ($genau) {
  $format = "%.4f";
} else {
  $format = "%.2f";
}
printf($format, $ergebnis);
```

Tipp

Das Listing *printf.php* enthält alle Beispiele zu `printf()` aus diesem Abschnitt.

`printf()` könnte man deswegen auch gut verwenden, um eine Ergebnismeldung unterschiedlich – je nach gewählter Sprache – auszugeben.

Neben `printf()` gibt es auch `sprintf()`. `sprintf()` funktioniert genauso wie `printf()`, aber es erlaubt die Speicherung des formatierten Strings in einer Variable:

```
$formergebnis = sprintf($format, $ergebnis);
```

Bisher haben Sie `%s` für String, `%d` für Integer und `%f` für eine Fließkommazahl kennen gelernt. Alle möglichen Zeichenkombinationen listet die folgende Tabelle auf.

String	Bedeutung
`%%`	Prozentzeichen
`%b`	Binärzahl
`%c`	entsprechendes ASCII-Zeichen
`%d`	Integer mit möglichen Vorzeichen
`%e`	Zahl in wissenschaftlicher Notation
`%u`	positiver vorzeichenloser Integer

Tabelle 6.1: Mögliche Formatierungsstrings bei printf()

String	Bedeutung
%f	Float – abhängig von Einstellung in Locales (siehe Abschnitt 6.4.2) unterschiedliche Dezimaltrennzeichen
%F	Float – unabhängig von Einstellung der Locales
%o	Oktalzahl
%s	String
%x	Hexadezimalwert in Kleinbuchstaben
%X	Hexadezimalwert in Großbuchstaben

Tabelle 6.1: Mögliche Formatierungsstrings bei printf() (Forts.)

Zahlen formatieren

Im Deutschen ist es üblich, als Tausendertrennzeichen den Punkt zu benutzen und als Dezimaltrennzeichen das Komma. Diese Transformation lässt sich bequem per `number_format()` vornehmen. `number_format()` erwartet als ersten Parameter die zu formatierende Zahl und als zweiten Parameter die Anzahl an Dezimalstellen. Außerdem können Sie dann noch das Trennzeichen für die Nachkommastellen und das Tausendertrennzeichen bestimmen.

In Listing 6.6 wird eine den deutschsprachigen Gegebenheiten entsprechende Zahl ausgegeben.

Listing 6.6: Eine Zahl wird so formatiert, wie es im deutschsprachigen Raum üblich ist (number_format.php).

```
$erg = 12345.678;
echo number_format($erg, 2, ",", ".");
```

Dies gibt aus: 12.345,68.

6.2.2 Suchen, Finden und Ersetzen

Viele Funktionen helfen dabei, wenn Sie prüfen möchten, ob ein Teilstring in einem String enthalten ist, oder auch Zeichen ersetzen möchten.

`strpos()` ermittelt das erste Vorkommen eines Zeichens in einem String. Die Zählung beginnt bei 0. `strpos()` erwartet zwei Parameter: Übergeben Sie als ersten Parameter, worin Sie suchen möchten, und als zweiten, was Sie suchen möchten. Es liefert `false` zurück, wenn nichts gefunden wird.

```
$satz = "Der Hund hat einen Knochen";
$suche = "noch";
$pos = strpos($satz, $suche);
if ($pos === false) {
```

```
  echo "$suche ist nicht in $satz";
} else {
  echo "$suche befindet sich an Position $pos in '$satz'";
}
```

Im Beispiel wird nach »noch« innerhalb des Satzes »Der Hund hat einen Knochen« gesucht und da es gefunden wird, ausgegeben, an welcher Stelle es sich befindet.

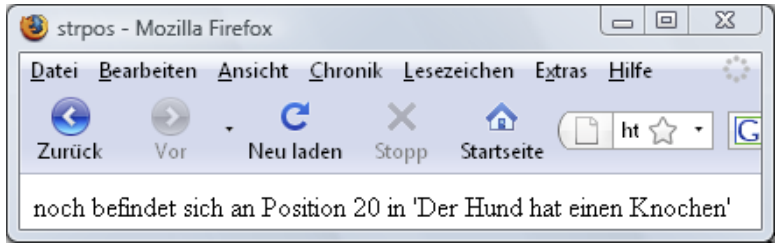


Abbildung 6.4: noch wurde im Satz gefunden.

Im Beispiel wird für die Überprüfung, ob der String vorhanden ist, das dreifache `=` verwendet, das gleichzeitig auch überprüft, ob die beiden Operanden vom selben Typ sind. Der Grund dafür: Angenommen, `$pos` liefert `0` zurück, weil der String an der `0`. Position zu finden ist. Dann würde `$pos == false` ebenfalls `true` zurückliefern, da `0`, in einen Booleschen Wert konvertiert, `false` ergibt. Durch das `===` erhalten Sie auch in diesem Fall das richtige Ergebnis. Testen können Sie dies, wenn Sie einmal nach »Der« in dem Beispielstring suchen – nur mit `===` erhalten Sie das erwartete Ergebnis.

Listing 6.7: Finden, wo etwas ist (strpos.php)

```
echo "<br />\n";
$satz = "Der Hund hat einen Knochen";
$suche = "Der";
$pos = strpos($satz, $suche);
if ($pos === false) {
    echo "$suche ist nicht in $satz";
} else {
    echo "$suche befindet sich an Position $pos in '$satz'";
}
```

`substr()` ist eine weitere praktische Funktion: Sie dient dazu, Teile aus Strings zu extrahieren. Bei `substr()` geben Sie zuerst den String an, den Sie zerlegen möchten, dann von wo Sie beginnen möchten. Der dritte Parameter ist fakultativ. Wenn Sie nichts angeben, wird der String bis zum Ende zurückgeliefert, ansonsten können Sie hier auch die Anzahl von Zeichen bestimmen, die Sie haben möchten.

Listing 6.8: Teile ausschneiden (substr.php)

```
01 $satz = "Der Hund hat einen Knochen";
02 $ausschnitt = substr($satz, 4, 8);
03 echo $ausschnitt;
04 echo "<br />\n";
05 $gekuerzt = substr($satz, 0, 12);
06 echo $gekuerzt;
07 echo "<br />\n";
08 $ende = substr($satz, -13);
09 echo $ende;
10 echo "<br />\n";
11 $kuerzer = substr($satz, 0, -7);
12 echo $kuerzer;
```

`substr()` ist ebenfalls gut geeignet, um einen längeren String auf eine vordefinierte Anzahl von Zeichen zu kürzen: Dann ist die Startposition 0 und als Länge geben Sie die Anzahl an Zeichen an, die Sie erhalten möchten. Wenn Sie `substr()` einen negativen Wert für die Startposition übergeben wie in Zeile 8, wird von hinten gezählt. Bei einem negativen Wert für Länge (Zeile 11) wird die entsprechende Anzahl an Zeichen entfernt.

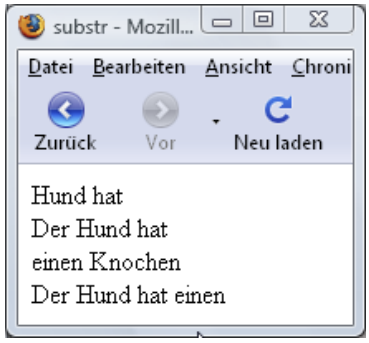


Abbildung 6.5: Mit substr() lassen sich Teile aus Strings ausschneiden.

`str_replace()` ersetzt im angegebenen String bestimmte Zeichenfolgen durch andere.

Listing 6.9: Einfache Ersetzung (str_replace.php)

```
$satz = "außerhalb der Straße";
$erg  = str_replace("ß", "ss", $satz);
echo "Vorher: $satz, nachher: $erg";
echo "<br />\n";
$text = "Schöne Grüße";
$sonder = array("ö", "ä", "ü", "ß", "Ö", "Ü", "Ä");
$ohne = array("oe", "ae", "ue", "ss", "Oe", "Ue", "Ae");
echo str_replace($sonder, $ohne, $text);
```

Im Beispiel wird `str_replace()` zweimal eingesetzt: Beim ersten Mal wird *ß* durch *ss* ersetzt. Beim zweiten Beispiel werden Arrays benutzt, sowohl für die Zeichen, die ersetzt werden sollen, als auch für die Zeichen, durch die sie ersetzt werden sollen: In diesem Fall wird das erste Zeichen aus dem ersten Array durch das erste Zeichen aus dem zweiten Array ersetzt usw.

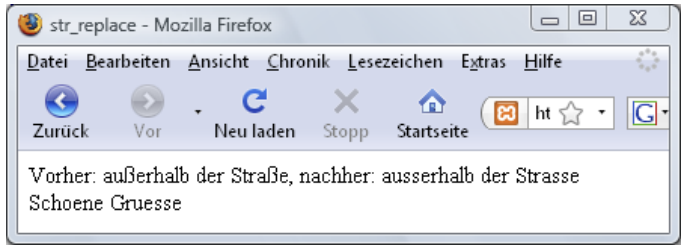


Abbildung 6.6: Ersetzung durch str_replace()

Eine recht praktische Funktion ist `explode()`. `explode()` verwandelt einen String in ein Array, indem es den String an einem vorgegebenen Zeichen aufteilt. Zwei Parameter erwartet `explode()`: zuerst das Trennzeichen, an dem die Aufteilung stattfinden soll, und dann den String, der aufgeteilt werden soll. Zurückgeliefert wird ein Array (Listing 6.10).

Listing 6.10: String in Bestandteile zerlegen über explode() (explode.php)

```
$str = "Müsli Rosinen Bananen";
$liste = explode(" ", $str);
print_r($liste);
```

Im Beispiel wird der String am Leerzeichen aufgeteilt. Das Array mit den einzelnen Elementen wird dann per `print_r()` ausgegeben.

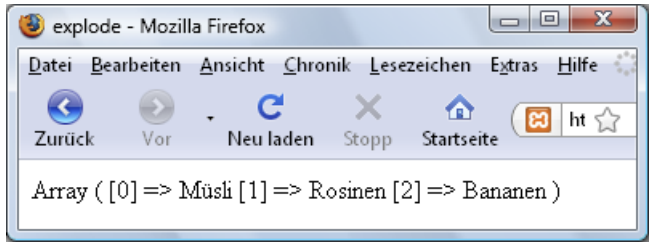


Abbildung 6.7: Ein Array als Ergebnis

6.2.3 Volle Freiheit mit regulären Ausdrücken

`str_replace()` ist eine ganz wunderbare Funktion, um einzelne Zeichenfolgen durch andere zu ersetzen. Aber nicht immer ist alles so einfach: Nicht immer suchen Sie

nach einem genau definierten Ausdruck, sondern manchmal nach einem Ausdruck, der einem *bestimmten Schema* (Muster) folgt, um diesen rauszufiltern, zu überprüfen oder zu ersetzen. Telefonnummern etwa folgen einem bestimmten Schema – es gibt nur Zahlen, Klammern, Bindestriche und Schrägstriche und keine anderen Zeichen. Genau das können reguläre Ausdrücke: die Mustererkennung.

Hinweis

Im Folgenden erfahren Sie, wie Sie Perl-kompatible reguläre Ausdrücke (Perl compatible regular expression = PCRE) erstellen. Daneben kann man in PHP auch reguläre Ausdrücke nach dem POSIX-Standard verwenden. Die Funktionen für die POSIX-kompatiblen beginnen mit `ereg`, die Perl-kompatiblen mit `preg`. Die POSIX-kompatiblen Ausdrücke sind aber seit PHP 5.3 unerwünscht und bei ihrer Verwendung erhält man eine DEPRECATED-Meldung. Deswegen wird auf die POSIX-kompatiblen Ausdrücke samt der zugehörigen Funktionen hier nicht weiter eingegangen.

Aufbau von regulären Ausdrücken

Sehen wir uns einen ersten regulären Ausdruck an. Die Perl-kompatiblen regulären Ausdrücke werden mit Begrenzungszeichen begrenzt, hier wird üblicherweise / eingesetzt.

Das Folgende wäre ein einfaches Muster, das für die Zeichenfolge *und* steht.

```
$muster = "/und/";
```

Jetzt brauchen wir eine Testmöglichkeit, um zu ermitteln, ob ein bestimmter regulärer Ausdruck auf einen String passt. Dafür kann man `preg_match()` einsetzen. Es erwartet als ersten Parameter das Muster und als zweiten den String, der überprüft werden soll. Wenn das Muster auf den String passt, liefert `preg_match()` 1 zurück, wenn nicht 0. Das lässt sich gut in eine `if`-Verzweigung einbauen, denn 1 wird zu `true` ausgewertet, 0 zu `false`:

Listing 6.11: Prüfen, ob ein Suchmuster in einem String vorhanden ist (preg_match.php)

```
$muster = "/und/";
$in = "Der Hund hat einen Knochen";
if(preg_match($muster, $in)) {
  echo "$muster passt auf '$in'";
}
```

In diesem Beispiel wird ausgegeben, dass das Muster auf den String passt, denn die Zeichenfolge *und* ist im String vorhanden.

Nun genauer zum Aufbau von Mustern: Die meisten Zeichen stehen für sich selbst, so wie im Beispiel `/und/` für die Buchstabenfolge *und* steht. Mächtig und interessant werden die regulären Ausdrücke durch verschiedene Sonderzeichen.

`.` – der Punkt – steht für ein beliebiges Zeichen: `/.und/` passt auf *Hund*, aber auch auf *Mund* oder *kund*. Wollen Sie einen Punkt hingegen wörtlich verwenden, müssen Sie ihn maskieren, d.h. mit einem Backslash schützen `\.`.

Sie können Muster »verankern«. Wenn Sie wollen, dass der Treffer am Anfang des Strings liegen muss, brauchen Sie `^`, wenn Sie wollen, dass der Treffer am Ende des Strings liegen soll, benutzen Sie das `$`-Zeichen. `^` und `$` lassen sich gleichzeitig verwenden:

- `/^.und/` passt nicht auf *Der Hund*, aber auf *Hund*.
- `/^.er/` passt auf *Der Hund*.
- `/.chen$/` passt auf *Knochen*.
- `/^hat$/` passt auf *hat*, aber nicht auf *hatte* oder auf *anhat*.

Wenn an einer Stelle eines von mehreren Zeichen stehen kann, können Sie dies durch Zeichenklassen angeben. Zeichenklassen werden in eckigen Klammern aufgeführt:

`/[HM]und/` passt auf *Hund* oder *Mund*, nicht aber auf *Fund*.

Soll keines der in eckigen Klammern angegebenen Zeichen vorkommen, schreiben Sie am Anfang ein `^`-Zeichen:

`/[^HM]und/` passt auf *Fund*, *kund*, aber nicht auf *Hund* oder *Mund*.

Alternativen legen Sie über die Pipe `|` fest.

`/Jacke|Hose/` passt auf *Jacke* oder *Hose*. Aber natürlich auch auf *Hosenträger*, denn es ist nicht verankert,

Anstelle von `/[HM]und/` können Sie auch `/(H|M)und/` schreiben, die Klammern dienen hierbei zur Gruppierung. Außerdem können die Klammern benutzt werden, um einen Teilbereich eines Suchmusters zu markieren – um es später wiederzuverwenden. Dazu gleich mehr.

Ein *Zeichenbereich* kann über einen Bindestrich festgelegt werden:

- `/[A-Z]und/` passt auf einen beliebigen Großbuchstaben, gefolgt von *und*.
- `/[0-9]-mal/` passt auf *1-mal*, *2-mal*, aber nicht auf *dreimal*.

Über Quantifizierer spezifizieren Sie, wie oft ein bestimmtes Zeichen vorkommen darf:

- `/^[A-Za-z]+und$/` passt auf *Stund*, *Hund* und auch auf *Baskettballbund*.
- `/https?:/` passt auf *http:* oder *https:*.

Quantifizierer	Bedeutung
`+`	bedeutet, dass das Zeichen/die Zeichenklasse mindestens einmal, aber beliebig häufig vorkommen darf
`?`	steht für ein- oder keinmal
`*`	steht für keinmal bis beliebig oft
`{4}`	Genau 4-mal. Sie können statt `4` beliebige Zahlen angeben.
`{1,5}`	steht für mindestens einmal, höchstens fünfmal. Anstelle von 1 und 5 sind beliebige Zahlen möglich. Sie können auch nur einen Minimalwert angeben: `{7,}` bedeutet, mindestens 7-mal bis zu beliebig viele. Umgekehrt bedeutet `{,7}` höchstens 7-mal.

Tabelle 6.2: Quantifizierer bei regulären Ausdrücken

Sollen sich Quantifizierer auf mehrere Zeichen beziehen, setzen Sie Klammern ein: `/^(http:\/\/)?/` passt auf *http://php.net*, aber auch auf *php.net*.

Tipp

Im Beispiel müssen Sie den / durch den Backslash maskieren, weil als Begrenzungszeichen ebenfalls der / benutzt wird und der PHP-Interpreter sonst beim zweiten Vorkommen von / das Ende des regulären Ausdrucks annimmt. Sie könnten sich die Maskierung auch sparen, wenn Sie andere Begrenzungszeichen benutzen, beispielsweise das @-Zeichen: `@^(http://)?@`.

Zeichenklassen haben Sie oben kennen gelernt. Für manche gibt es Abkürzungen:

Ausdruck	Bedeutung
`\s`	Whitespace (Leerzeichen, Newlinezeichen, Tabulator etc.)
`\S`	Nicht-Whitespace-Zeichen
`\b`	Wortgrenze
`\B`	Nicht-Wortgrenze
`\d`	Ziffer
`\D`	Nicht-Ziffer
`\w`	Wortzeichen (Ziffern, Buchstaben sowie Unterstrich)
`\W`	Nicht-Wortzeichen

Tabelle 6.3: Ausdrücke für bestimmte Klassen von Zeichen

`/\bnoch\b/` passt nicht auf *Knochen*, aber auf *Das auch noch!*.

Tipp

Die anderen \-Sequenzen – also `\n` für einen Zeilenumbruch oder `\t` für einen Tabulator (siehe Kapitel 4) – gelten natürlich auch weiterhin.

Soll die Suche unabhängig von Groß- und Kleinschreibung funktionieren, benutzen Sie `i` als Modifizierer nach dem Begrenzungszeichen für den regulären Ausdruck:

- `@http://@i` passt auch auf *HTTP://PHP.NET*.

Modifizierer	Bedeutung
`i`	Groß- und Kleinschreibung wird ignoriert.
`m`	Nur relevant, wenn das Suchmuster `^` und `$` sowie die Zeichenkette `\n` beinhaltet. Mit diesem Modifizierer passen Zeilenanfang- und Zeilenende-Konstrukte dann nicht nur auf den Anfang und das Ende der Zeichenkette, sondern auch auf den Anfang und das Ende nach oder vor einem Zeilenumbruch.
`s`	Ist diese Option gesetzt, passt der Punkt (`.`) auch auf das Newline-Zeichen.
`U`	Bewirkt, dass Quantifizierer standardmäßig nicht gierig sind (Beispiel dazu weiter unten)

Tabelle 6.4: Modifizierer und ihre Bedeutung

Achtung

Viele Zeichen haben bei regulären Ausdrücken eine Sonderbedeutung, wie beispielsweise der Punkt, das Sternchen oder das Fragezeichen. Denken Sie daran, diese zu maskieren, d.h. mit einem Backslash zu versehen, wenn Sie sie in Ihrer wörtlichen Bedeutung benutzen wollen. Allerdings haben die meisten Zeichen ihre Sonderbedeutung nicht innerhalb von eckigen Klammern. Innerhalb von eckigen Klammern ist ein Punkt ein Punkt.

Treffer zurückgeben lassen

Noch einmal zurück zur Funktion `preg_match()`. In Listing 6.11 haben Sie gesehen, dass man sie verwenden kann, um festzustellen, ob ein Muster auf einen String passt. `preg_match()` erwartet als ersten Parameter das Muster und als zweiten den String, in dem nach dem Muster gesucht werden soll. `preg_match()` erlaubt noch einen dritten optionalen Parameter: Wenn Sie hier eine Variable übergeben, werden Ihnen darin die Treffer geliefert.

Listing 6.12: Treffer ausgeben lassen (preg_match_treffer.php)

```
$muster = "/und/";
$in = "Der Hund hat einen Knochen";
if(preg_match($muster, $in, $treffer)) {
  echo "$muster passt auf '$in'<br />\n";
  print_r($treffer);
}
```

Der Inhalt des Arrays wird über `print_r()` ausgegeben: Es besteht im Beispiel aus einem Arrayelement mit dem Treffer.

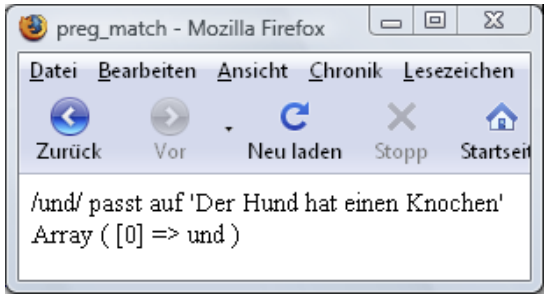


Abbildung 6.8: Die Treffer werden als Array zurückgegeben.

Wenn Sie im Suchmuster mit runden Klammern arbeiten, können Sie auf deren Inhalte über weitere Arrayelemente zugreifen. Ein Beispiel dazu sehen Sie bei `preg_match_all()`.

Hinweis

Falls Sie das Verhalten des dritten Parameters etwas seltsam finden: Die Erklärung dafür ist, dass dieser Parameter per Referenz übergeben wird und damit in der Funktion geändert werden kann. Im PHP-Manual (*http://www.php.net/manual/de/function.preg-match.php*) erkennen Sie das daran, dass bei der Funktionsbeschreibung beim Parameter der `&`-Operator steht.

Manchmal möchte man auch mehrere Vorkommen eines Suchmusters innerhalb eines Strings finden. Hierfür gibt es die Funktion `preg_match_all()`, die genauso wie `preg_match()` funktioniert, aber *alle Treffer* findet.

Listing 6.13: Mehrere Treffer finden (preg_match_all.php)

```
$muster = "/\b\d{3}\b/";
$str = "123 456 7890 123";
if(preg_match_all($muster, $str, $treffer)) {
```

```
  echo "$muster passt auf '$str'<br />\n";
  print_r($treffer);
}
```

Im Beispiel sollen Dreiergruppen von Zahlen gefunden werden. Im Suchmuster wird zuerst eine Wortgrenze verlangt `\b`, dann drei Ziffern `\d{3}` und wieder eine Wortgrenze `\b`. Gefunden werden drei Treffer, das Array `$treffer` wird per `print_r()` angezeigt.

Sie sehen, dass das Array `$treffer` ein zweidimensionales Array ist. Über `$treffer[0][0]` könnte man auf den ersten Treffer zugreifen (*123*), über `$treffer[0][1]` auf den zweiten (*456*).

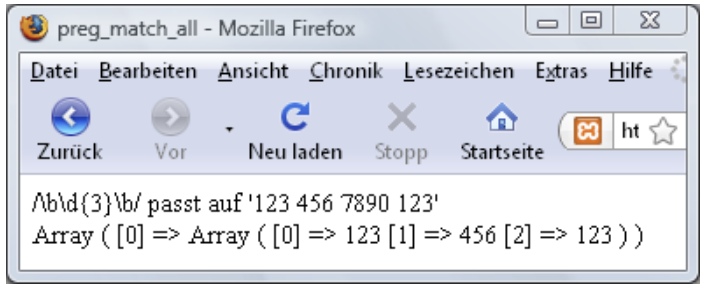


Abbildung 6.9: Drei Treffer werden angezeigt.

Inhalte eines (X)HTML-Elements auslesen

Noch ein weiteres Beispiel für einen regulären Ausdruck. Oft möchte man auf ein bestimmtes (X)HTML-Element zugreifen und den Inhalt auslesen. Versuchen wir einmal einen regulären Ausdruck zu schreiben, der die Inhalte von `p`-Elementen ausliest.

Aus folgendem (X)HTML-Ausschnitt wollen wir die Inhalte haben, also »Ein Absatz und das geht gleich weiter«, »Hier folgt der zweite« sowie »Und ein dritter« (im Folgenden fett hervorgehoben):

```
<p class='example'>Ein Absatz und das geht gleich weiter</p><p>Hier folgt der
zweite</p><p>Und ein dritter</p>
```

Nun zum Muster. Da wir innerhalb des Musters selbst das `/` brauchen, empfiehlt es sich, ein anderes Begrenzungszeichen zu nehmen, beispielsweise das `@`-Zeichen. Dann sieht ein erster Entwurf des regulären Ausdrucks so aus:

```
$muster = "@<p[^>]*>.*</p>@";
```

Was wir suchen, beginnt mit `<p`, dann können Attribute (z.B. oben `class='example'`) folgen, bevor eine spitze schließende Klammer `>` kommt. Wie lassen sich die möglichen Attribute über einen regulären Ausdruck definieren? Es kann ein Attribut sein, aber es können auch mehrere sein oder aber keins. Eine Möglichkeit wäre jetzt zu über-

legen, welche Zeichen hier vorkommen können. Die andere Möglichkeit ist es, davon auszugehen, dass das, was dann folgt, auf jeden Fall keine schließende Klammer ist `[^>]`. Und von diesen »Nicht-spitze-Klammer-Zeichen« können 0 bis beliebig viele kommen, das kennzeichnet das *. Dann kommt die schließende spitze Klammer. Das, was innerhalb des Elements `p` steht, kann ganz unterschiedlich sein, auch ein leerer Absatz ist denkbar – dafür wird `.*` benutzt. Was jetzt folgt ist einfach – das schließende `p`-Tag `</p>`. Hier stehen alle Zeichen für sich und haben keine Sonderbedeutung.

Fügen wir das Muster in die Testumgebung ein, um das Ergebnis zu sehen:

```
$muster = "@<p[^>]*>.*</p>@";
$in = "<p class='example'>Ein Absatz und das geht gleich weiter</p><p>Hier folgt der
zweite</p><p>Und ein dritter</p>";
if(preg_match_all($muster, $in, $treffer)) {
  echo "$muster passt auf '$in'<br />\n";
  print_r($treffer);
}
```

Das Ergebnis sieht man besser, wenn man in die Quellcode-Ansicht im Browser wechselt (Abbildung 6.10). Das Erstaunliche: Es wurden nicht wie angenommen, drei Treffer, sondern nur einer gefunden.

```
Array
(
    [0] => Array
        (
            [0] => <p class='example'>Ein Absatz und das geht gleich weiter</p><p>Hier folgt der zweite</p><p>Und ein dritter</p>
        )

)
```

Abbildung 6.10: Nur ein Treffer – das war anders gedacht.

Der reguläre Ausdruck passt auf den Gesamtstring. Warum denn das? Das liegt an der *Gefräßigkeit der Quantifizierer*, am Ausdruck `.*`. Dieser Ausdruck besagt ja, dass ein beliebiges Zeichen folgt und davon 0 bis unendlich viele. Im Beispiel nimmt dieser Ausdruck alles von der spitzen schließenden Klammer vom Starttag von `p` einschließlich des Worts »dritter« – also bis die spitze Klammer vom Endtag beginnt. Das ist das Standardverhalten von Quantifizierern: Sie sind gierig, und nehmen so viel sie kriegen können. In diesem Fall ist das Verhalten nicht erwünscht. Um die Quantifizierer dazu zu bringen, nicht so viel wie möglich, sondern *so wenig wie nötig* zu nehmen, benutzen Sie den Modifizierer `U`.

```
$muster = "@<p[^>]*>.*</p>@U";
```

Mit dieser Modifikation passt es wie gewünscht – das Array mit den Treffern enthält nun drei Elemente. (Warum wir nur den Inhalt der Absätze erhalten? Dazu kommen wir gleich.)

```
Array
(
    [0] => Array
        (
            [0] => <p class='example'>Ein Absatz und das geht gleich weiter</p>
            [1] => <p>Hier folgt der zweite</p>
            [2] => <p>Und ein dritter</p>
        )

)
```

Abbildung 6.11: Drei Treffer

Noch eine weitere Modifikation: Wenn ein Zeilenumbruch innerhalb eines Absatztextes steht, klappt das Auslesen nicht.

```
$in = "<p class='example'>Ein Absatz
       und das geht gleich weiter</p><p>Hier folgt der zweite</p><p>Und ein
dritter</p>";
```

Denn der Punkt . gilt für ein beliebiges Zeichen, nicht aber das Newline-Zeichen. Wieder gibt es einen nützlichen Modifizierer, der dies ändert: das kleine `s`:

```
$muster = "@<p[^>]*>.*</p>@Us";
```

Noch eine letzte Sache. Wir wollten ja den Inhalt von Absätzen auslesen und nicht die gesamten Absätze mit Textinhalt erhalten. Dafür setzen wir runde Klammern um den Inhalt, den wir auslesen wollen.

```
$muster = "@<p[^>]*>(.*)</p>@Us";
```

Das Array `$treffer` beinhaltet dann als erstes Element ein Array mit den Treffern, die auf den gesamten Ausdruck passen. Das zweite Element des Arrays ist das Array mit den Treffern für den geklammerten Ausdruck.

Damit lassen sich die einzelnen Textinhalte der Absätze aus dem Array `$treffer[1]` auslesen.

Tipp

Wenn Sie mehrere Klammernpaare benutzen, werden weitere Arrayelemente für diese Treffer bereitgestellt.

```
Array
(
    [0] => Array
        (
            [0] => <p class='example'>Ein Absatz
       und das geht gleich weiter</p>
            [1] => <p>Hier folgt der zweite</p>
            [2] => <p>Und ein dritter</p>
        )

    [1] => Array
        (
            [0] => Ein Absatz
       und das geht gleich weiter
            [1] => Hier folgt der zweite
            [2] => Und ein dritter
        )

)
```

Abbildung 6.12: Das erste Arrayelement enthält die Treffer für das Gesamtmuster, das zweite die Treffer für den Ausdruck in runden Klammern.

Hier noch einmal das fertig gestellte Beispiel:

Listing 6.14: So klappt es mit dem Auslesen des Inhalts der `p`-Elemente (reg_exp_beispiel.php).

```
$muster = "@<p[^>]*>(.*)</p>@Us";
$in = "<p class='example'>Ein Absatz
       und das geht gleich weiter</p><p>Hier folgt der zweite</p><p>Und ein
        dritter</p>";
if(preg_match_all($muster, $in, $treffer)) {
  echo "passt <br />\n";
  print_r($treffer);
}
```

Sie haben Beispiele für die Verwendung von `preg_match()` und `preg_match_all()` gesehen. Weitere praktische Funktionen sind `preg_replace()` für Ersetzung auf der Basis von regulären Ausdrücken oder `preg_split()` zur Zerlegung von Zeichenketten anhand eines regulären Ausdrucks.

Allgemeine Tipps zur Verwendung von regulären Ausdrücken

Reguläre Ausdrücke sind sehr mächtig, aber auch relativ komplex und unhandlich. Prinzipiell gilt: Wenn Sie eine Aufgabe ohne reguläre Ausdrücke lösen können, dann lösen Sie sie ohne reguläre Ausdrücke. Feste Zeichenfolgen sucht man schneller mit `str_pos()` und den anderen in Abschnitt 6.2.2 vorgestellten Funktionen. Für viele Aufgaben liefert PHP auch schon selbst Funktionen – so könnte man mit regulären Ausdrücken die (X)HTML-Tags aus Strings herauslöschen –, besser erledigt es die Funktion `strip_tags()`. Es lohnt sich deswegen immer ein Blick in das Manual, ob es für den Zweck, den Sie brauchen, nicht vielleicht schon eine fertige Lösung gibt.

Wenn Sie einen regulären Ausdruck schreiben, sollten Sie ihn mit möglichst vielen möglichst unterschiedlichen Eingaben testen. Um zu überprüfen, ob etwas ein Nachname sein könnte, könnte man erst einmal vermuten, dass bei Namen nur Buchstaben erlaubt sind. Es ist neben Buchstaben aber auch das `'`-Zeichen möglich (*O'Brien*) und es gibt Doppelnamen mit oder ohne Bindestrich (*Müller-Thurgau*).

6.2.4 Zusammenarbeit mit (X)HTML

PHP wird üblicherweise zusammen mit (X)HTML eingesetzt, deswegen gibt es natürlich genau darauf spezialisierte Funktionen.

Die spitzen Klammern haben in (X)HTML eine besondere Bedeutung, denn dadurch werden Tags gekennzeichnet. Auch das `&`-Zeichen hat eine spezielle Funktion: Es dient zur Einleitung von Entities wie `ä`. Wenn Sie diese Zeichen nicht für Tags oder Entities benutzen wollen, sondern normal im Text ausgeben, müssen Sie stattdessen Entities verwenden. Dafür können Sie `htmlspecialchars()` benutzen. Die Funktion `htmlspecialchars()` wandelt alle (X)HTML spezifischen Sonderzeichen um. Dabei werden folgende Ersetzungen durchgeführt.

- `&` wird zu `&`
- `<` wird zu `>`
- `>` wird zu `>`
- `"` wird zu `"`

`htmlspecialchars()` erwartet als ersten Parameter Text, der behandelt werden muss. Über einen zweiten Parameter können Sie die Behandlung von Anführungszeichen über Konstanten festlegen. Diese können folgende Werte haben:

- `ENT_QUOTES`: Beide Anführungszeichen werden umgewandelt, d.h. zusätzlich zu den oben geschilderten auch einfache Anführungszeichen.
- `'` wird zu `'`
- `ENT_NOQUOTES`: Keines der Anführungszeichen wird umgewandelt.
- `ENT_COMPAT` ist der Standard: Die doppelten Anführungszeichen werden transformiert, die einfachen nicht.

Das folgende Listing führt den Einsatz von `htmlspecialchars()` mit den verschiedenen Optionen vor. Außerdem wird die Funktion `htmlspecialchars_decode()`eingesetzt, die die Umwandlung von `htmlspecialchars()` rückgängig macht. Sie steht erst seit PHP 5.1 zur Verfügung.

Listing 6.15: htmlspecialchars() wandelt (X)HTML-Sonderzeichen um (htmlspecialchars.php).

```
01 $behauptung = "2 < 4 && \"5\" == '5'";
02 echo htmlspecialchars($behauptung);
03 echo "<br />\n";
```

```
04 echo htmlspecialchars($behauptung, ENT_QUOTES);
05 echo "<br />\n";
06 echo htmlspecialchars($behauptung, ENT_NOQUOTES);
07 echo "<br />\n";
08 $mit = htmlspecialchars($behauptung);
09 $ohne = htmlspecialchars_decode($mit);
10 echo $ohne;
```

Das Ergebnis zeigt Abbildung 6.13. Hier sehen Sie den (X)HTML-Quellcode, die Entities sind deutlich zu sehen.

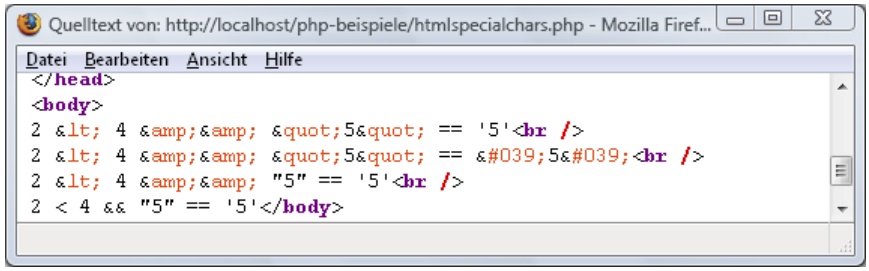


Abbildung 6.13: htmlspecialchars() und zurück

Die Funktion `htmlentities()` ist ähnlich wie `htmlspecialchars()`. Sie wandelt aber zusätzlich zu den bei `htmlspecialchars()` umgewandelten Zeichen auch Sonderzeichen wie ä, ö und ähnliche Zeichen in die entsprechenden (X)HTML-Entities um.

Die Funktion `strip_tags()` geht einen Schritt weiter: Sie entfernt (X)HTML-Tags aus einem String. Als ersten Parameter geben Sie den String an, den Sie von Tags bereinigen möchten, als zweiten Parameter können Sie (X)HTML-Elemente spezifizieren, die nicht entfernt werden sollen:

Listing 6.16: (X)HTML-Tags entfernen (strip_tags.php)

```
01 $string = "<p>Ein Absatz mit <em>mehrfacher</em> <strong>Betonung</strong></p>";
02 echo $string;
03 echo "\n";
04 echo strip_tags($string);
05 echo "\n";
06 echo strip_tags($string, "<p><em>");
```

Der behandelte String enthält verschiedene Tags: `<p>`, `<strong>` und `<em>` (Zeile 1). Gibt man `strip_tags()` ohne zweiten Parameter an, werden einfach alle Tags rausgelöscht (Zeile 4). Beim zweiten Aufruf von `strip_tags()` wird hingegen festgelegt, dass `<p>` und `<em>` stehen bleiben sollen (Zeile 4). In diesem Fall wird nur `<strong>` entfernt.

`htmlspecialchars()` und `strip_tags()` spielen eine entscheidende Rolle bei der Absicherung Ihrer Skripten gegen bösartige Manipulationen (Kapitel 7).

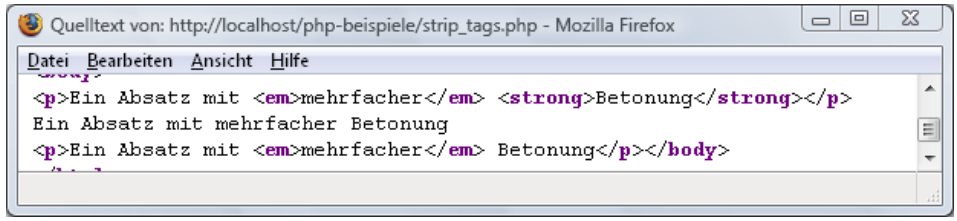


Abbildung 6.14: Tags rauslöschen

Hinweis

Auch für die Bearbeitung von URLs gibt es eigene Funktionen: Um selbst Parameter über URLs zu übergeben, können Sie `urlencode()` verwenden. Sonderzeichen werden über `%` mit zwei hexadezimalen Zahlen kodiert, Leerzeichen als `+`. `urldecode()` macht das wieder rückgängig. Ebenfalls interessant ist `parse_url()`, über die Sie eine URL in ihre Bestandteile zerlegen können.

6.2.5 Zeichenkodierungen

In Kapitel 3 haben Sie gesehen, wie Sie im (X)HTML-Dokument die Kodierung des Dokuments festlegen – über die `meta`-Angabe. Dies ist der richtige Moment, um das etwas genauer zu betrachten.

Kodierungen

Es gibt unterschiedliche Kodierungen.

Der ASCII-Code umfasst beispielsweise 128 verschiedene Zeichen. Um jedes eindeutig darzustellen, genügen 7 Bit.

Bei den ISO-Zeichensätzen wird dem 7-stelligen Code eine Stelle hinzugefügt, sodass 256 Zeichen kodiert werden können, wofür ein 8 Bit (ein Byte) benötigt wird. Die ersten 127 Zahlencodes der ISO-Zeichensätze gehören dem ASCII-Basisalphabet, die Codes von 128 bis 255 werden bei jedem ISO-Zeichensatz anders vergeben. Für das Deutsche verwendet werden ISO-8859-1 oder ISO-8859-15, wobei Letzterer zusätzlich das Eurozeichen enthält.

Exkurs: Zeichenkodierung angeben

Die Zeichenkodierung bestimmen Sie über verschiedene Methoden:

- Über eine `meta`-Angabe im `head`-Bereich des Dokuments selbst:

```
<meta http-equiv="content-type" content="text/html; charset=ISO-
8859-1" />
```

- Bei XHTML auch über die XML-Deklaration, die zu Beginn eines Dokuments steht:

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  ```

  Diese ist jedoch aus zwei Gründen problematisch. Erst einmal gibt es einen Bug im Internet Explorer 6, die bewirkt, dass er bei Vorhandensein der XML-Deklaration sich verhält wie die Vorversionen (Internet Explorer 5.x) und damit Webseiten wesentlich anders darstellt als andere moderne Browser.

  Zudem kommen Sie in die Bredouille, wenn die Short Open Tag-Option in PHP aktiviert ist, dann wird nämlich `<?xml` als Anfang von PHP-Code missverstanden. Umgehen können Sie dieses Problem, indem Sie die XML-Deklaration mit `echo` über PHP ausgeben lassen:

  ```
  echo "<?xml version='1.0' encoding='ISO-8859-1'?>";
  ```

- Über einen HTTP-Header, den der Server mitsendet:

  ```
  Content-Type: text/html;charset=ISO-8859-1
  ```

  Dies können Sie beispielsweise über eine *.htaccess*-Datei machen (mehr zu *.htaccess*-Dateien im Anhang) oder über die `header`-Funktion von PHP (siehe Kapitel 8).

  Wichtig ist jedoch, dass die vom Server gesendete Header-Information sich im Zweifelsfall durchsetzt, d.h. dass diese gilt, auch wenn im Dokument ein abweichender Zeichensatz angegeben ist.

Die ISO-Zeichensätze sind beschränkt in ihrem Einsatzbereich: Zum einen gibt es natürlich Sprachen, die nicht mit den in ISO frei verfügbaren (die ersten 127 Zeichen sind ja immer gleich) 128 Zeichen auskommen. Und zum anderen können Sie immer nur einen Zeichensatz pro Dokument verwenden. Die ISO-Zeichensätze werden im Web viel verwendet, aber selbst nicht mehr weiterentwickelt. Anders ist das bei Unicode, der im Web hauptsächlich als UTF-8 zum Einsatz kommt

Unicode hat zum Ziel, für alle Zeichen aller bekannten Schriftkulturen einen digitalen Code festzulegen. Bei UTF-8 wird der alte 7-Bit-ASCII-Code nicht nur wie bei den ISO-Zeichensätzen um 1 Bit aufgestockt, sondern es werden gleich zwischen 1 und 4 Byte für jedes Zeichen zur Verfügung gestellt. Die ersten 128 Zeichen sind identisch und durch 1 Byte dargestellt, für die anderen Zeichen werden mehr Bytes zur Verfügung gestellt.

Probleme bei UTF-8

Wie schaut es jetzt mit PHP und den verschiedenen Zeichenkodierungen aus? Sie können UTF-8 benutzen, Text über PHP ausgeben lassen, das funktioniert problemlos. Jedoch sind die normalen Stringbearbeitungsfunktionen von PHP auf die Verwendung von 1-Byte-Zeichensätzen ausgerichtet. Das heißt: Wenn Sie UTF-8 einsetzen, kann es zu unerwarteten Ergebnissen kommen. Ein Beispiel dafür ist die Funktion `strlen()`:

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
06   <title>strlen mit UTF-8</title>
07 </head>
08 <body>
09 <?php
10 $str = "Mäßigung";
11 $anz = strlen($str);
12 echo "'$str' hat $anz Zeichen";
13 ?>
14 </body>
15 </html>
```

In diesem Beispiel wird ein String »Mäßigung« definiert, die Anzahl von Zeichen ermittelt und ausgegeben. Zählen Sie einmal kurz selbst. *Mäßigung* hat 8 Zeichen – nicht wahr? PHP sieht das anders (Abbildung 6.15).

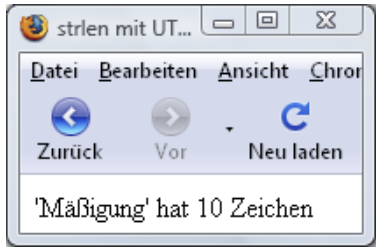


Abbildung 6.15: Komische Art zu zählen ...

Wie Sie in Abbildung 6.15 sehen, ermittelt PHP als Anzahl 10 Zeichen. Der Grund dafür: PHP geht von einem 1-Byte-Zeichensatz aus und die Nicht-ASCII-Zeichen `ä` und `ß`, die über 2-Byte kodiert werden, bringen die Berechnungen aus dem Tritt.

Man kann natürlich hier tricksen, damit es wieder funktioniert: PHP stellt die praktische Funktion `utf8_decode()` zur Verfügung, die einen UTF-8 kodierten String in einen ISO-8859-1-kodierten umwandelt. Wenn man diese aufruft, und vom Ergebnis die Anzahl an Zeichen zählen lässt, dann klappt alles wieder.

Listing 6.17: strlen() bei einem UTF-8-kodierten Dokument (strlen_utf8.php)

```
$anz = strlen(utf8_decode($str));
echo "'$str' hat $anz Zeichen";
```

Jetzt wird die korrekte Anzahl – 8 Zeichen – ausgegeben.

Tipp

Parallel zu `utf8_decode()` gibt es `utf8_encode()`, das einen ISO-8859-1 kodierten String in UTF-8 umwandelt. Beides sehr nützliche Funktionen beim Einsatz von UTF-8!

Ein weiteres Beispiel für Probleme mit UTF-8 sind die Funktionen `strtoupper()`, um alle Buchstaben eines Strings in Großbuchstaben zu verwandeln, und `strtolower()`, zur Umwandlung in Kleinbuchstaben.

`strtoupper()` und `strtolower()` funktionieren problemlos, wenn Sie sie mit 1-Byte-Zeichensätzen nutzen. Bei UTF-8 hingegen kann es wieder zu Problemen kommen, wie folgendes Beispiel zeigt.

Listing 6.18: Wechsel zwischen Groß- und Kleinbuchstaben (gross_klein_mehrbyte.php)

```
01 $str = " Münchner Hofbräuhaus<br />\n";
02 echo $str;
03 echo strtoupper($str);
04 echo strtolower($str);
05 echo "<br />----mit Multibyte-Funktion ---<br />\n";
06 echo $str;
07 echo mb_strtoupper($str, "UTF-8");
08 echo mb_strtolower($str, "UTF-8");
```

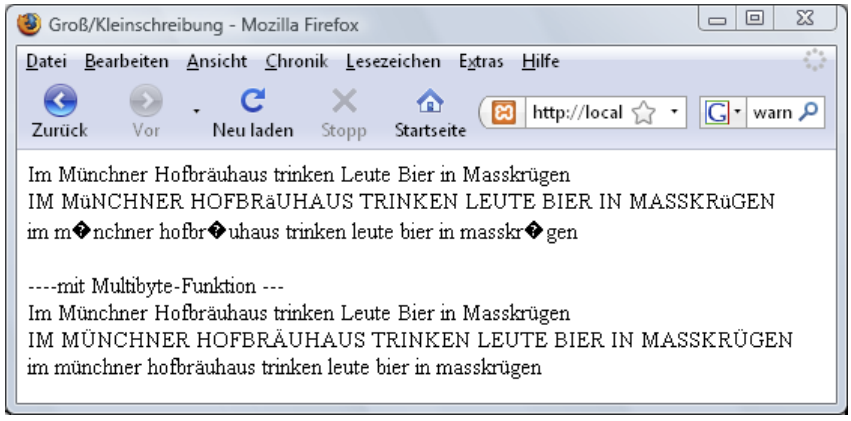


Abbildung 6.16: Ungereimtheiten bei der Verwendung von strtolower() und strotoupper() mit der Kodierung UTF-8. Mit den Multibyte-Entsprechungen geht's hingegen.

Auch in diesem Dokument wird UTF-8 benutzt. Ein String wird definiert und dann mit `strtoupper()`in Großbuchstaben bzw. mit `strtolower()` in Kleinbuchstaben ausgegeben. Dass die Ausgabe nicht wie gewünscht funktioniert, zeigt Abbildung 6.16.

In Zeile 7 und 8 werden zwei andere Funktionen – die so genannten Multibyte-Varianten dieser Funktionen – eingesetzt, bei denen man zusätzlich die benutzte Kodierung angibt. Jetzt klappt die Ausgabe wie gewünscht.

Hinweis

Damit Sie die Multibyte-Funktionen einsetzen können, muss diese Erweiterung *mbstring* installiert sein. Ob das der Fall ist, erfahren Sie über die Ausgabe von `phpinfo()`. Suchen Sie hier nach dem Abschnitt *mbstring*.

Dieses etwas mühselige Handling bei UTF-8 wird sich ändern: PHP bringt in Version 6 eine *native Unterstützung für Unicode* und dann gehören diese Probleme der Vergangenheit an.

6.3 Funktionen für Arrays

Auch für Arrays gibt es viele nützliche Funktionen, um die es in diesem Abschnitt geht.

6.3.1 Arrays und Strings

In Abschnitt 6.2.2 haben Sie gesehen, wie man per `explode()` aus einem String ein Array macht. Genau das Umgekehrte macht `implode()`: Es verbindet mehrere Arrayelemente zu einem String. Es erwartet als ersten Parameter einen String, der die einzelnen Elemente zusammenkleben soll, und als zweiten das Array:

Listing 6.19: Arrayelemente zu einem String verketten über implode() (implode.php)

```
$liste = array ("Oliven", "Kapern", "Ananas");
$str = implode(" und ", $liste);
echo $str;
```

`implode()` ist eine gute Funktion, wenn Sie einen String erhalten möchten, den Sie beispielsweise auch Benutzern präsentieren können. Soll der Inhalt von Arrays hingegen nur aus programmiertaktischen Gründen in einen String umgewandelt werden mit dem Ziel, das später rückgängig zu machen, so sind die Funktionen `serialize()` und `unserialize()` das Richtige.

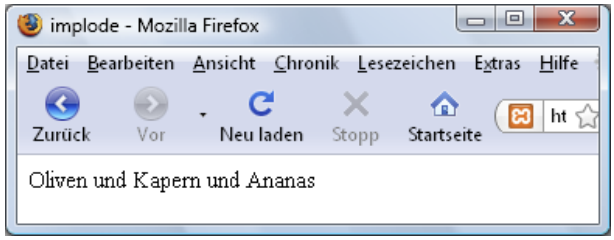


Abbildung 6.17: Der aus den Arrayelementen zusammengesetzte String

Tipp

Das könnte man beispielsweise brauchen, um mehrere Werte aus einem Array in einem Cookie zu speichern (siehe Kapitel 8), die Informationen aus einem Array über ein verstecktes Feld (siehe Kapitel 7) weiterzugeben oder ein Array in einer Datenbanktabelle zu speichern (Kapitel 11).

Listing 6.20: Ein Array wird serialisiert (serialisieren.php).

```
$farben = array ("hellblau" =>  "lightblue",
                 "schwarz" => "black",
                 "gelb" => "yellow",
                 "himmelblau" => "skyblue",
                 "rot" => "red");
$serial = serialize($farben);
echo $serial;
$arr = unserialize($serial);
print_r($arr);
```

Im Beispiel wird ein assoziatives Array erstellt, serialisiert und ausgegeben. Nach der Deserialisierung sieht es genauso aus wie zuvor.

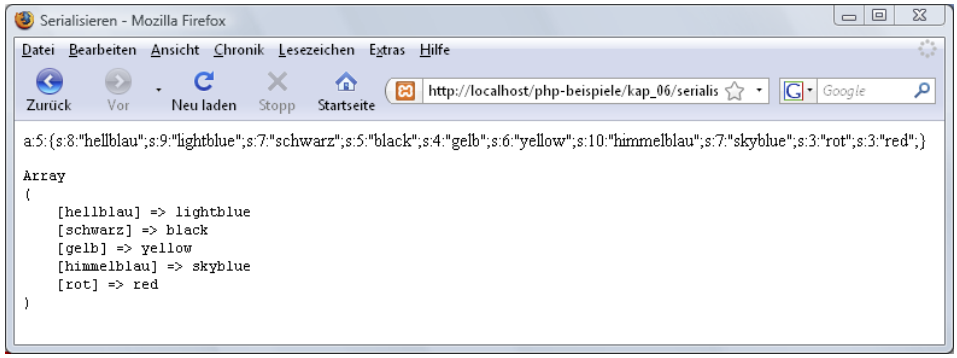


Abbildung 6.18: Einmal Serialisieren und zurück

6.3.2 Arrays sortieren

Viele unterschiedliche Funktionen dienen zum Sortieren von Arrays – Sie können bei Arrays sowohl Schlüssel als auch Werte sortieren und das noch auf- oder absteigend etc.

Für indizierte Arrays geeignet sind `sort()` und `rsort()`. Beide erwarten als Parameter ein Array, das per Referenz übergeben wird. Das bedeutet: Die Funktionen ändern das Array selbst.

Tipp

Im Manual erkennen Sie das wieder daran, dass bei der Funktionsbeschreibung vor `$array` der Referenzoperator steht:

```
bool sort  ( array &$array  [, int $sort_flags  ] )
```

`sort()` sortiert aufsteigend, `rsort()` absteigend.

Listing 6.21: Array sortieren (sortieren.php)

```
echo "<pre>";
$liste = array ("Kapern", "Oliven", "Ananas");
sort($liste);
print_r($liste);
rsort($liste);
print_r($liste);
echo "</pre>";
```

Tipp

Wenn Sie Umlaute oder andere sprachspezifische Sonderzeichen in Ihren Strings verwenden, so funktioniert die Sortierung nicht wie gewünscht. Damit das funktioniert, müssen Sie über `setlocale()` die Sprache bestimmen und bei `sort()` zusätzlich `SORT_LOCALE_STRING` angeben:

Listing 6.22: So werden die deutschen Umlaute richtig einsortiert (sortieren_sprachspezifisch.php).

```
$umlaute = array ("Abend", "Oper", "Öl", "Ärger");
setlocale(LC_ALL, "de_DE@euro", "de_DE", "deu_deu");
sort($umlaute, SORT_LOCALE_STRING);
print_r($umlaute);
```

Diese Sortierfunktionen sind für indizierte Arrays geeignet, für assoziative Arrays brauchen Sie andere Funktionen, bei denen der Zusammenhang zwischen Schlüssel und Wert beibehalten wird.

`asort()` macht genau das: Es sortiert ein Array nach Werten und behält die Verbindung zum Index. `ksort()` sortiert entsprechend nach den Schlüsseln. Von beiden gibt es noch die Rückwärtsvariante: `arsort()` sortiert ein Array nach Werten absteigend und `krsort()` sortiert absteigend nach Schlüsseln.

Listing 6.23: Sortieren von assoziativen Arrays (assoziative_sortieren.php)

```
01 $farben = array ("hellblau" =>  "lightblue",
02                  "schwarz" => "black",
03                  "gelb" => "yellow",
04                  "himmelblau" => "skyblue",
05                  "rot" => "red");
06 echo "<strong>asort()</strong>: ";
07 asort($farben);
08 print_r($farben);
09 echo "<br /><strong>arsort()</strong>: ";
10 arsort($farben);
11 print_r($farben);
12 echo "<br /><strong>ksort()</strong>: ";
13 ksort($farben);
14 print_r($farben);
15 echo "<br /><strong>krsort()</strong>: ";
16 krsort($farben);
17 print_r($farben);
```

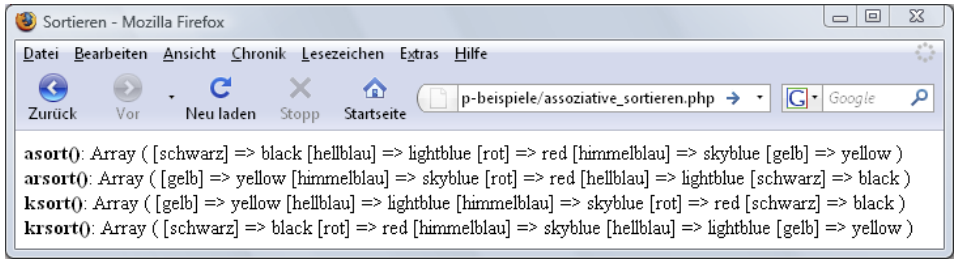


Abbildung 6.19: Zuerst wird vorwärts und rückwärts nach Werten sortiert, dann nach Schlüssel: Die Zuordnung von Wert zu Schlüssel bleibt dabei erhalten.

6.3.3 Weitere Arrayfunktionen

Um die Elemente von Arrays einzelnen Variablen zuzuweisen, ist das Sprachkonstrukt `list()` richtig.

Listing 6.24: Die Elemente eines Arrays werden einzelnen Variablen zugewiesen (list.php).

```
$liste = array ("Kapern", "Oliven", "Ananas");
list($a, $b, $c) = $liste;
echo "mit $a, $b und $c ...";
```

`array_map()`, das dazu dient, eine Operation bei jedem einzelnen Arrayelement durchzuführen, kennen Sie bereits aus Kapitel 4 aus den Beispielen für anonyme Funktionen. Neben benutzerdefinierten Funktionen können auch von PHP bereitgestellte Funktionen angegeben werden. Im folgenden Beispiel wird `trim` bei allen Arrayelementen angewandt.

Listing 6.25: Alle Arrayelemente werden mit trim() behandelt (array_map.php).

```
$liste = array (" Kapern ", "  Oliven ", "  Ananas ");
foreach($liste as $el) {
  echo "'$el'";
}
$liste = array_map("trim", $liste);
echo "<br />\n nach trim <br />\n";
foreach($liste as $el) {
  echo "'$el'";
}
```

```
' Kapern ''  Oliven ''  Ananas '<br />
 nach trim <br />
'Kapern''Oliven''Ananas'
```

Abbildung 6.20: Der Ausschnitt aus dem Quellcode zeigt die Arrayelemente vor der array_map() mit vielen Leerzeichen und nach der Anwendung von array_map() ohne Leerzeichen.

Um zu ermitteln, ob ein Array bestimmte Elemente enthält, dienen `in_array()` und `array_search()`. `in_array()` liefert `true` zurück, wenn ein bestimmter Wert in einem Array enthalten ist, `array_search()` liefert in diesem Fall hingegen den Schlüssel des Arrays zurück.

Beide Funktionen erwarten als ersten Parameter den Wert, der im Array gesucht werden soll. Als zweiten Parameter geben Sie das Array an, in dem gesucht werden soll. Optional können Sie als dritten Parameter `true` angeben, dann wird auch überprüft, ob die Datentypen übereinstimmen.

Listing 6.26: Bestimmen, ob ein Wert in einem Array vorhanden ist (array_werte_suchen.php)

```
$farben = array ("hellblau" =>  "lightblue",
                 "schwarz" => "black",
                 "gelb" => "yellow",
                 "himmelblau" => "skyblue",
                 "rot" => "red");
```

```
$such = "yellow";
if (in_array($such, $farben)) {
  echo "$such vorhanden. Zugehöriger Schlüssel: ";
  echo array_search($such, $farben);
}
```

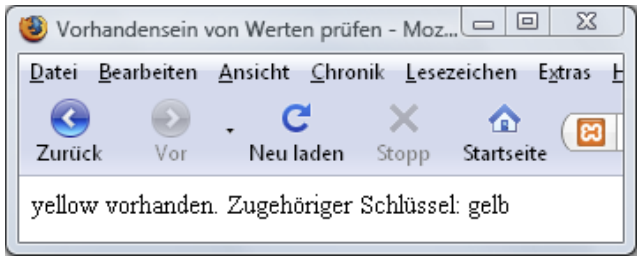


Abbildung 6.21: Der Wert yellow wurde gefunden und der zugehörige Schlüssel ermittelt.

Hinweis

Dies ist nur ein kleiner Ausschnitt aus den Arrayfunktionen von PHP. Weitere finden Sie im Manual unter *http://de.php.net/manual/de/ref.array.php*.

6.4 Arbeiten mit Datum und Uhrzeit

Dass man per PHP das Datum ausgeben kann, haben Sie schon an einigen Beispielen gesehen. Nun erfahren Sie genauer, wie `date()` und andere Datumsfunktionen funktionieren und welche Optionen sie bieten.

6.4.1 Datum formatiert ausgeben über date()

Es gibt mehrere Funktionen, um ein aktuelles Datum formatiert auszugeben. Eine davon ist `date()`. Sie erwartet als ersten Parameter einen Formatierungsstring. Mit diesem geben Sie an, welche Informationen (Jahr, Monat, Tag, Stunde oder/und Minute etc. etc.) und auf welche Art Sie diese erhalten möchten. So kann man beispielsweise bestimmen, ob man den Monat als Zahl oder als englische Bezeichnung haben möchte etc. Als zweiten Parameter können Sie einen Zeitstempel angeben (dazu mehr in Abschnitt 6.4.3). Wenn Sie hier nichts angeben, wird das aktuelle Datum genommen.

Die folgende Tabelle führt die wichtigsten Formatierungsoptionen auf:

Formatierungsstring	Erklärung	Beispiel
Tag		
d	Zweistelliger Tag des Monats	01 bis 31
D	Abgekürzter Wochentag	Mon bis Sun
j	Tag des Monats ohne führende 0	1 bis 31
l	Wochentag ausgeschrieben	Sunday bis Saturday
N	Eine der ISO-8601-Norm entsprechende Darstellung des Wochentags (erst seit PHP 5.1.0)	1 (Montag) bis 7 (Sonntag)
w	Numerische Darstellung des Wochentags	0 (Sonntag) bis 6 (Samstag)
z	Der wie vielte Tag des Jahres es ist – die Zählung beginnt bei 0	0 bis 365
Woche		
W	Gibt an, die wie vielte Kalenderwoche im Jahr es ist. Begonnen wird mit Montag. Ist konform zur ISO-8601-Norm.	10
Monat		
F	Ausgeschriebener Monatsname (englisch)	January bis December
m	Monat zwischen 01 und 12	01 bis 12
M	Abgekürzter Monatsname	Jan bis Dec
n	Monat zwischen 1 und 12 (ohne führende 0)	1 bis 12
t	Anzahl der Tage in einem Monat	28 bis 31
Jahr		
L	Schaltjahr	1, wenn Schaltjahr, sonst 0
Y	Vierstellige Jahreszahl	2009
y	Zweistellige Jahreszahl	99 oder 08
Zeit		
a	Kleingeschrieben am oder pm	am oder pm
A	AM oder PM großgeschrieben	AM oder PM
g	Stunde von 1 bis 12 ohne führende 0	1 bis 12
G	Stunde von 0 bis 23 ohne führende 0	0 bis 23
h	Stunde von 1 bis 12 mit führender 0	01 bis 12
H	Stunde von 0 bis 23 mit führender 0	00 bis 23
i	Minuten mit führender 0	00 bis 59
s	Sekunden mit führender 0	00 bis 59
u	Millisekunden (seit PHP 5.2.2)	54321

Tabelle 6.5: Mögliche Angaben bei date()

Formatierungsstring	Erklärung	Beispiel
Zeitzone		
e	Identifiziert die Zeitzone (seit PHP 5.1.0)	UTC, GMT, Atlantic/Azores
I (großes i)	Sommerzeit oder nicht	Bei Sommerzeit, 1, sonst 0
O	Unterschied zur mittleren Greenwich-Zeit (GMT) in Stunden	+0200
P	Unterschied zur GMT in Stunden mit Doppelpunkt zwischen Stunden und Minuten (seit PHP 5.1.3)	+02:00
T	Abkürzung der Zeitzone	EST, MDT ...
Z	Zeitzonenoffset relativ zur UTC	-43200 bis 50400
Vollständige Datums/Zeitangabe		
c	ISO-8601-Datum (seit PHP 5)	2009-04-27T08:30:21+02:00
r	Gemäß RFC 2822 formatiertes Datum	Thu, 21 Dec 2000 16:01:07 +0200
U	Seit der Unix-Epoche verstrichene Sekunden (1.1.1970 00:00:00 GMT)	

Tabelle 6.5: Mögliche Angaben bei date() (Forts.)

Tipp

Wenn Sie `date()` einsetzen, müssen Sie die Defaultzeitzone setzen, sonst erhalten Sie eine Warnung. In PHP-Versionen vor 5.3 wird eine Meldung nach dem Strict Standard angezeigt.

Im Beispiel wird als Zeitzone immer »Europe/Berlin« verwendet. Weitere mögliche stehen online unter *http://de.php.net/manual/de/timezones.php*.

Das folgende Skript zeigt den Einsatz von `date()`.

Listing 6.27: Datum, Uhrzeit und Zusatzinformationen ausgeben lassen per date() (date.php)

```
01 date_default_timezone_set("Europe/Berlin");
02 echo date("d.m.Y H:i:s");
03 echo "<br />\n";
04 echo date("d.m.Y, \u\m H  \U\h\\r i");
05 echo "<br />\nDer aktuelle Monat hat ";
06 echo date("t") . " Tage.<br />\n";
07 if (date("I") == 1) {
```

```
08    echo "Es ist Sommerzeit.";
09 } else {
10   echo "Es ist keine Sommerzeit.";
11 }
```

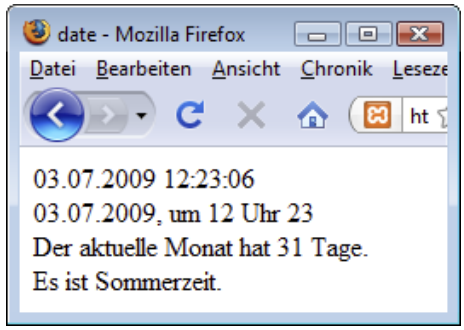


Abbildung 6.22: Über date() formatierte Ausgaben

In der ersten Zeile wird die Defaultzeitzone gesetzt, dann kommt `date()` zum Einsatz. Beim ersten Aufruf von `date()` in Zeile 2 werden Formatierungsstrings mit Satzzeichen kombiniert – mit Punkten und Doppelpunkten, wie man es gewohnt ist. Bei der zweiten Verwendung von `date()` (Zeile 4) wird zusätzlicher Text ausgegeben (`um` und `Uhr`). Dabei müssen Sie Zeichen, die innerhalb von `date()` eine spezielle Bedeutung haben, mit einem `\` maskieren. Sonderfall ist das `r`. Hier genügt ein Backslash nicht, da `\r` für einen Wagenrücklauf steht (siehe auch Kapitel 4). Deswegen werden in diesem Fall zwei `\` benutzt.

Nun folgen zwei weitere nützliche Informationen, die sich per `date()` ermitteln lassen. In Zeile 6 wird ermittelt, wie viele Tage der aktuelle Monat hat. In Zeile 7 bis 11 wird überprüft, ob gerade Sommerzeit ist oder nicht und eine entsprechende Meldung ausgegeben.

`date()` gibt Wochentage und Monatsnamen in Englisch aus. Um diese einzudeutschen, können Sie ein Array mit den deutschsprachigen Werten übergeben und dann immer den entsprechenden auslesen.

Listing 6.28: Deutsche Wochentage und Monatsnamen (date_deutsch.php)

```
01 date_default_timezone_set("Europe/Berlin");
02 $tage = array (
03         "Mon" => "Montag",
04         "Tue" => "Dienstag",
05         "Wed" => "Mittwoch",
06         "Thu" => "Donnerstag",
07         "Fri" => "Freitag",
08         "Sat" => "Samstag",
09         "Sun" => "Sonntag"
```

```
10         );
11 $monate = array (
12         "Jan" => "Januar",
13         "Feb" => "Februar",
14         "Mar" => "März",
15         "Apr" => "April",
16         "Mai" => "Mai",
17         "Jun" => "Juni",
18         "Jul" => "Juli",
19         "Aug" => "August",
20         "Sep" => "September",
21         "Oct" => "Oktober",
22         "Nov" => "November",
23         "Dec" => "Dezember"
24         );
25 $monat = $monate[date("M")];
26 $wochentag = $tage[date("D")];
27 echo "$wochentag, den ";
28 echo date("j. " ) .$monat . date(" Y");
```

Im Listing werden zwei assoziative Arrays angelegt: Im Array `$tage` (Zeile 2) wird den englischen kurzen Wochentagen immer der entsprechende deutsche Wochentag zugeordnet. Das Array `$monate` macht dasselbe für die Monate. In Zeile 25 wird der aktuelle deutsche Monat ermittelt: Verwendet wird hierfür das Array `$monate`, dem als Schlüssel die englische Kurzform des Monats übergeben wird. Genau das ermittelt `date("M")`. In Zeile 26 wird dasselbe für den Wochentag durchgeführt und dann das Datum ausgegeben.

Abbildung 6.23: Eine mögliche Ausgabe von date() nun mit deutschen Wochentagen und Monat

Dieses Beispiel ließe sich natürlich auch objektorientiert erstellen: Über eine Klasse namens `Datum`, die die Methode `ausgeben()` enthält:

Listing 6.29: Dieses Mal objektorientiert (date_deutsch_oo.php)

```
01 class Datum
02 {
03   public function ausgeben()
```

```
04   {
05     date_default_timezone_set("Europe/Berlin");
06     $tage = array (
07         "Mon" => "Montag",
08         "Tue" => "Dienstag",
09         "Wed" => "Mittwoch",
10         "Thu" => "Donnerstag",
11         "Fri" => "Freitag",
12         "Sat" => "Samstag",
13         "Sun" => "Sonntag");
14     $wochentag = $tage[date("D")];
15     $monate = array (
16         "Jan" => "Januar",
17         "Feb" => "Februar",
18         "Mar" => "März",
19         "Apr" => "April",
20         "Mai" => "Mai",
21         "Jun" => "Juni",
22         "Jul" => "Juli",
23         "Aug" => "August",
24         "Sep" => "September",
25         "Oct" => "Oktober",
26         "Nov" => "November",
27         "Dec" => "Dezember");
28    $monat = $monate[date("M")];
29    $wochentag = $tage[date("D")];
30    echo "$wochentag, den ";
31    echo date("j. " ) .$monat . date(" Y");
32   }
33 }
34 $heute = new Datum();
35 $heute->ausgeben();
```

Damit dann auch wirklich etwas ausgegeben wird, wird in Zeile 34 ein neues Objekt zur Klasse `Datum` erstellt und die Methode `ausgeben()` aufgerufen.

6.4.2 strftime() und setlocale()

Eine weitere Möglichkeit, Datum und Uhrzeit formatiert auszugeben, ist über `strftime()`. `strftime()` erwartet wie `date()` als ersten Parameter einen Formatierungsstring. Mit diesem spezifizieren Sie, welche Information in welchem Format ausgegeben werden soll. Im Unterschied zu `date()` werden die Formatierungszeichen aber mit einem `%`-Zeichen davor gekennzeichnet. Ein weiterer, wesentlicher Unterschied zu `date()` ist, dass `strftime()` die Werte unterschiedlich von den gesetzten Locale-Informationen ausgibt. Das bedeutet, wenn Sie die Locale auf Deutsch setzen, erhalten Sie *Mittwoch* als Wochentag und nicht *Wednesday*.

Zur Setzung der Locale-Information dient die Funktion `setlocale()`. `setlocale()` setzt die Gebietsschemaparameter, d.h. den lokalen Gegebenheiten entsprechende Zahlen-, Währungs-, Datums- und Zeitformate. Die exakte Verwendung von `setlocale()` ist je nach benutztem Betriebssystem unterschiedlich und außerdem muss, damit `setlocale()` funktionieren kann, die entsprechende Sprachunterstützung auch auf dem System installiert sein.

Durch folgenden String werden verschiedene mögliche Locale-Namen für Deutsch probiert:

```
setlocale(LC_ALL, "de_DE@euro", "de_DE", "deu_deu");
```

Es gibt jedoch eine Einschränkung bei der Benutzung von `setlocale()`, es ist nicht thread-sicher. Im PHP-Manual steht dazu folgende Warnung: »Die Locale-Informationen wirken auf den Prozess, nicht auf den Thread. Sofern Sie PHP mit einer Multithreaded Server API wie IIS oder Apache unter Windows einsetzen, rechnen Sie mit unerwarteten Änderungen der Locale-Einstellungen zur Laufzeit des Skripts, auch wenn das Skript selbst keinen `setlocale()`-Aufruf durchführt.« (*http://de.php.net/manual/de/function.setlocale.php*).

Ein Beispiel hierzu: Zuerst werden die Locale-Informationen gesetzt, dann verschiedene Datumsangaben ausgegeben. Da Prozentzeichen zur Kennzeichnung der Formatierungsstrings benutzt werden, lässt sich `strftime()` problemlos mit der Ausgabe von weiterem Text kombinieren, wie Sie in Listing 6.30 sehen.

Listing 6.30: Einsatz von strftime() (strftime.php)

```
date_default_timezone_set("Europe/Berlin");
setlocale(LC_ALL, "de_DE@euro", "de_DE", "deu_deu");
echo strftime("Heute ist %A.<br />\n");
echo strftime("Monat: %B<br />\n");
echo strftime("Datum: %x<br />\n");
echo strftime("Uhrzeit: %X<br />\n");
```

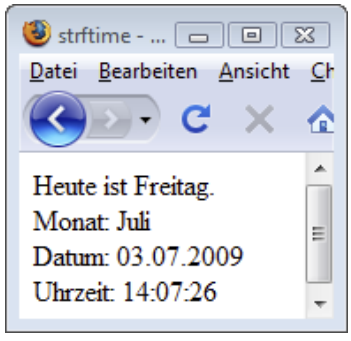


Abbildung 6.24: Lokalisierte Ausgabe von strftime()

Die folgende Tabelle führt wichtige mögliche Angaben von `strftime()` auf:

Formatierungsstring	Bedeutung
`%a`	Wochentag, abgekürzt
`%A`	Wochentag, ausgeschrieben
`%b`	Monat, abgekürzt
`%B`	Monat, ausgeschrieben
`%c`	Datum und Zeit
`%x`	Datum
`%X`	Zeit

Tabelle 6.6: Ausgewählte Formatierungsangaben für strftime()

Eine vollständige Liste der Formatierungsangaben für `strftime()` finden Sie online unter *http://www.php.net/manual/de/function.strftime.php*.

Tipp

Beachten Sie aber beim Ausprobieren, dass nicht alle Formatierungsangaben auch unter Windows funktionieren. Welche unter Windows funktionieren, finden Sie, wenn Sie bei *http://msdn.microsoft.com/* im Suchfeld `strftime` eingeben.

6.4.3 Ein beliebiges Datum festlegen

Bisher haben Sie mit `strftime()` und `date()` zwei Möglichkeiten gesehen, um das *aktuelle* Datum auszugeben. Sie können aber auch ein anderes Datum als zweiten Parameter in Form eines Zeitstempels übergeben. PHP rechnet intern mit dem Zeitstempel, das sind die seit der Unix-Zeit – dem 1. Januar 1970 00:00 Uhr UTC – vergangenen Sekunden.

Achtung

Der gültige Bereich eines Zeitstempels liegt typischerweise zwischen dem 13. Dezember 1901 und dem 19 Januar 2038.

Den aktuellen Zeitstempel können Sie mit `time()` ausgeben lassen. Wenn Sie den Zeitstempel eines bestimmten Datums erzeugen wollen, verwenden Sie dazu `mktime()`. `mktime()` erwartet als Parameter – in dieser Reihenfolge – Stunde, Minute, Sekunde, Monat, Tag, Jahr.

Im folgenden Skript wird über `mktime()` der Zeitstempel vom 24.12.2009 um 17 Uhr erstellt und dieser dann über `date()` formatiert ausgegeben.

Listing 6.31: Mit mktime() kann man beliebige Zeitstempel erstellen (weihnachten.php).

```
01 date_default_timezone_set("Europe/Berlin");
02 $tage = array (
03         "Mon" => "Montag",
04         "Tue" => "Dienstag",
05         "Wed" => "Mittwoch",
06         "Thu" => "Donnerstag",
07         "Fri" => "Freitag",
08         "Sat" => "Samstag",
09         "Sun" => "Sonntag");
10 $weihnachten = mktime(17, 0, 0, 12, 24, 2009);
11 $wochentag = $tage[date("D", $weihnachten)];
12 echo "$wochentag, den ";
13 echo date("d.m.Y H:i:s", $weihnachten);
```

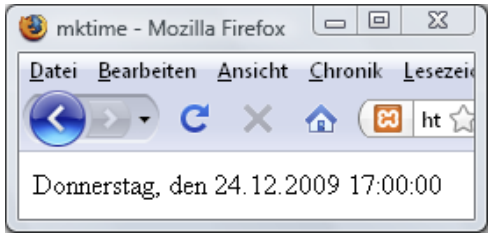


Abbildung 6.25: Weihnachten, genau genommen Heilig Abend, 2009 fällt auf einen Donnerstag.

`mktime()` in Kombination mit `date()` lässt sich auch nutzen, um ein relatives Datum zu ermitteln. Nehmen wir an, Sie möchten das Datum von vorgestern ausgeben.

```
$vorgestern  = mktime(0, 0, 0, intval(date("m")), intval(date("d")-2));
echo date("d.m.Y ", $vorgestern);
```

Im Beispiel bestimmen Sie mit `date("m")` den aktuellen Monat und über `date("d") - 2`, den aktuellen Tag minus 2. Das Jahr wird nicht angegeben und damit automatisch auf das aktuelle Jahr gesetzt. `intval()` wird hier zusätzlich benutzt, da `date()` einen String zurückgibt, `mktime()` aber eigentlich einen Integer erwartet. `intval()` macht aus dem String einen Integer – was der PHP-Interpreter ansonsten aber auch für Sie übernimmt.

Eine weitere Möglichkeit, ein relatives Datum zu ermitteln, ist über `strtotime()`. Dieses erwartet ein Datum entsprechend US-amerikanischer Schreibweise und gibt den dafür ermittelten Zeitstempel zurück. Erlaubt sind auch relative Angaben wie `+1 day/week/month/year` etc.:

Listing 6.32: Relative Angaben vom aktuellen Datum ab gerechnet, sind auch über strtotime() möglich (strtotime.php).

```
date_default_timezone_set("Europe/Berlin");
echo date("d.m.Y H:i:s", strtotime("+1 day"));
echo "<br />\n";
echo date("d.m.Y H:i:s", strtotime("+2 day"));
echo "<br />\n";
echo date("d.m.Y H:i:s", strtotime("+1 week"));
echo "<br />\n";
echo date("d.m.Y H:i:s", strtotime("next Monday"));
echo "<br />\n";
```

Je nachdem, wann Sie das Skript aufrufen, erhalten Sie natürlich ganz andere Ausgaben. Abbildung 6.25 zeigt das Ergebnis am 3. Juli 2009.

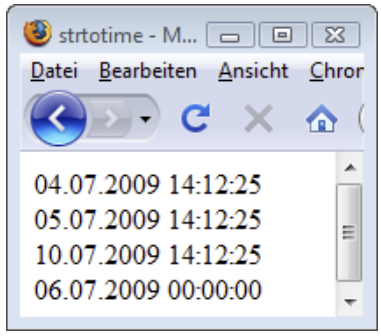


Abbildung 6.26: Basis für diese Ausgabe ist der 3. Juli 2009.

6.4.4 Differenz zwischen zwei Daten berechnen

Häufiger möchte man ermitteln, wie viel Zeit zwischen zwei Daten verstrichen ist. Das lässt sich relativ leicht erledigen, da Sie mit den Zeitstempeln zweier unterschiedlicher Daten rechnen können. Wenn Sie einen Zeitstempel von einem anderen subtrahieren, erhalten Sie die Differenz in Sekunden. Diese lassen sich dann bei Bedarf in eine andere Einheit wie Minuten, Stunden, Tage oder Wochen umrechnen. Genau das macht die Funktion `zeitdifferenz()` in folgendem Listing.

Listing 6.33: Berechnet den Unterschied zwischen zwei Zeitstempeln (zeitdifferenz.php)

```
01 date_default_timezone_set("Europe/Berlin");
02 function zeitdifferenz($t1, $t2, $einheit)
03 {
04   $differenz = abs($t1 - $t2);
05   $anzSek = array (
06     "Sekunden" => 1,
07     "Minuten"  => 60,
08     "Stunden"  => 3600,
09     "Tage"     => 86400,
10     "Wochen"   => 604800
11   );
12   if (isset($anzSek[$einheit])) {
13     return floor($differenz/$anzSek[$einheit]);
14   } else {
15     return "Ungültige Eingabe";
16   }
17 }
18 $vorgestern = mktime(0, 0, 0, intval(date("m")), intval(date("d")-2));
19 $heute      = time();
20 echo zeitdifferenz($heute, $vorgestern, "Tage");
21 echo "<br />\n";
22 echo zeitdifferenz($vorgestern, $heute, "Tage");
```

Die Funktion zur Berechnung der Differenz von zwei Zeitangaben erwartet drei Parameter: zwei Zeitstempel und außerdem die Einheit, in der das Ergebnis ausgegeben werden soll. In Zeile 4 wird die Differenz zwischen den zwei Zeitstempeln ermittelt. Falls `$t1` kleiner ist als `$t2`, wäre das Ergebnis negativ. Da eine Aussage wie »Zwischen X und Y liegen -2 Tage« eher ungewöhnlich wäre, wird zusätzlich die Funktion `abs()` eingesetzt. Diese liefert den Betrag ohne Vorzeichen, sorgt also dafür, dass das Ergebnis nicht negativ ist.

In Zeile 5 wird ein Array erstellt, das mögliche Einheiten möglichen Umrechnungswerten zuweist. Soll die Zeitdifferenz in Sekunden ausgegeben werden, so sind keine weiteren Umrechnungen möglich – der Faktor ist 1. Bei Minuten muss man das Ergebnis durch 60 teilen, bei Stunden durch 60*60 etc.

Zeile 12 überprüft, ob die angegebene Einheit stimmt, indem sie prüft, ob die Variable – das Arrayelement mit dem angegebenen Schlüssel – gesetzt ist. Ist das der Fall, wird die Umrechnung durchgeführt, d.h. die Differenz durch den je nach Einheit gewählten Faktor geteilt. `floor()` rundet das Ergebnis ab. Anstelle von 1.8 wird 1 ausgegeben – zurückgeliefert werden also nur die ganzen Tage/Wochen etc.

In Zeile 18 wird die Funktion getestet: Zuerst wird der Timestamp von vorgestern erstellt, dann der aktuelle mit `time()` ermittelt. Mit diesen beiden Parametern und der Einheit Tage wird die Funktion aufgerufen und das Ergebnis ausgegeben. Zeile 22

zeigt im Test, dass bei umgekehrter Reihenfolge (zuerst der Zeitstempel von vorgestern, dann der von heute) dasselbe Ergebnis rauskommt. Wie zu erwarten, wird beide Male »2« ausgegeben.

6.4.5 Datumsangabe überprüfen

Was Sie ebenfalls häufig brauchen werden, ist die Überprüfung einer Datumsangabe, um festzustellen, ob es das Datum überhaupt gibt. Genau das macht `checkdate()`. `checkdate()` berücksichtigt dabei auch Schaltjahre. Es erwartet drei Parameter: den Monat, den Tag und das Jahr. Es gibt `true` zurück, wenn das Datum existiert, und `false`, wenn nicht.

```
var_dump(checkdate(2, 29, 2008)); //true
var_dump(checkdate(2, 29, 2009)); //false
```

Falls Sie ein Datum überprüfen wollen, bei dem Sie die einzelnen Angaben von einem Benutzer erhalten haben, sollten Sie vorher mit `is_numeric()`prüfen, ob die Angaben numerisch sind. Die folgende Funktion `datumpruefen()` erwartet Tag, Monat und Jahr. Wenn Zahlen übergeben wurden, wird das Datum mit `checkdate()` überprüft und das Ergebnis zurückgeliefert. Ansonsten wird gleich `false` zurückgegeben.

Listing 6.34: Datum prüfen (checkdate.php)

```
01 function datumpruefen($t, $m, $j)
02 {
03   if (is_numeric($t) && is_numeric($m) && is_numeric($j)) {
04     return checkdate($m, $t, $j);
05   } else {
06     return false;
07   }
08 }
09 if (datumpruefen("a", 2, 2009)) {
10   echo "Termin gibt es";
11 } else {
12   echo "Termin gibt es nicht";
13 }
```

Im Beispiel sehen Sie dann, wie diese Funktion in einer `if`-Verzweigung eingesetzt wird. Da im Beispiel ein String »a« anstelle einer Zahl übergeben wurde, erscheint die Meldung »Termin gibt es nicht«.

6.5 Eindeutschen einfach gemacht über die Erweiterung intl

Sie haben an verschiedenen Stellen in diesem Kapitel gesehen, dass Sie sich bei der Eindeutschung (Lokalisierung) um bestimmte Dinge kümmern müssen: `sort()` sortiert erst einmal die deutschen Umlaute nicht richtig, und zur Ausgabe eines den

deutschen Gepflogenheiten entsprechenden Datums ist zusätzlicher Aufwand notwendig. Die an verschiedenen Stellen vorgestellte Möglichkeit, das über `setlocale()` zu machen, hat den Nachteil, dass `setlocale()` vom zugrunde liegenden Betriebssystem abhängig ist.

Das ist bei der in PHP 5.3 neuen Erweiterung `intl` besser.

Hinweis

Damit Sie diese Erweiterung verwenden können, brauchen Sie die ICU-Bibliothek (*http://site.icu-project.org/*), die bei PHP ab Version 5.3 dabei ist, und außerdem muss die Erweiterung aktiviert sein. Wenn das der Fall ist, finden Sie bei der Ausgabe von `phpinfo()` den entsprechenden Eintrag.

intl

Internationalization support	enabled
version	1.0.0
ICU version	3.6

Abbildung 6.27: Die intl-Erweiterung ist einsatzbereit.

Sehen wir uns an, wie man die `intl`-Erweiterung nutzen kann, um ein Array korrekt zu sortieren. Die Angabe der Sortierungsregeln nennt man *Kollationen*, englisch *Collator*. Um das Array zu sortieren, brauchen Sie zwei Funktionen: Zuerst muss `collator_create()` aufgerufen werden und ihr ein Locale-String für die gewünschte Sortierung übergeben werden. Dann kann das Array über `collator_sort()` sortiert werden. `collator_sort()` erwartet als ersten Parameter den Collator (Rückgabewert von `collater_create()`) und als zweiten Parameter das zu sortierende Array.

Listing 6.35: Sortieren mit der Erweiterung intl (intl_sort.php)

```
$umlaute = array ("Abend", "Oper", "Öl", "Ärger");
$coll = collator_create("de_DE");
collator_sort($coll, $umlaute);
print_r($umlaute);
```

Im Beispiel wird das sortierte Array zur Kontrolle am Schluss per `print_r()` ausgegeben.

Achtung

Setzt man die Erweiterung `intl` ein, muss als Zeichensatz UTF-8 verwendet werden.

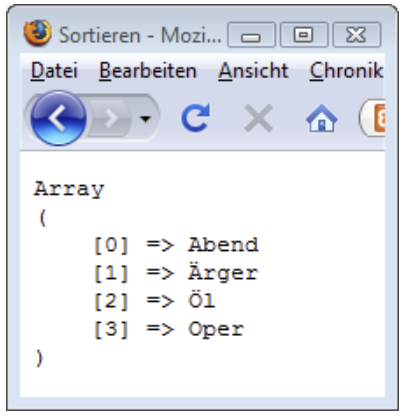


Abbildung 6.28: Richtig sortiert ...

Alternativ zu dieser prozeduralen Syntax, können Sie die Erweiterung auch objektorientiert verwenden. Das Beispiel lässt sich auch folgendermaßen schreiben:

Listing 6.36: Collator-Klasse einsetzen (intl_sortieren_oo.php)

```
$umlaute = array ("Abend", "Oper", "Öl", "Ärger");
$coll = new Collator("de_DE");
$coll->sort($umlaute);
print_r($umlaute);
```

Sie sehen: Es wird über `new Collator()` ein neues Collator-Objekt erstellt. Danach wird die Methode `sort()` des Collator-Objekts aufgerufen und ihr das zu sortierende Array übergeben.

Neben den Kollationen für die Sortierung gibt es weitere interessante Objekte, beispielsweise `NumberFormatter` für die Formatierung von Zahlen oder die `IntlDateFormatter`-Klasse zur Formatierung von Zahlen.

Tipp

Ausführliche Informationen bietet hier wie immer das PHP-Manual unter *http://www.php.net/manual/de/book.intl.php*.

Im nächsten Kapitel erfahren Sie, wie Sie in Formularen eingegebene Daten verarbeiten – und so mit Ihren Besuchern kommunizieren können.

7 Formulare verarbeiten mit PHP

Formulare sind eine zentrale Komponente von Webseiten, weil sie die komfortabelste Möglichkeit sind, Input von Benutzern zu erhalten: sei es das einfache Kontaktformular auf einer Webseite, ein Suchfeld zur site-internen Suche, ein Formular für einen Forenbeitrag oder das Formular bei einem Bestellvorgang im Online-Shop. (X)HTML stellt Ihnen die Formularfelder zur Verfügung – aber für die Weiterverarbeitung brauchen Sie PHP oder eine andere serverseitige Skriptsprache.

In diesem Kapitel erfahren Sie, wie Sie Formulare erstellen und mit PHP auf die Daten zugreifen. Ausführlich widmet sich das Kapitel dem Thema Sicherheit, das heißt, welche Gefahren durch bösartige Formulareingaben drohen und wie Sie Ihre Formulare absichern. Da die Daten aus Formularen häufig per Mail versendet werden, geht es in diesem Kapitel auch darum – sowohl um Text- als auch HTML-Mails. Im letzten Abschnitt dreht sich dann alles um den Upload von Dateien und konkret um das Hochladen und Speichern von Bildern.

7.1 Formularbasis

Abbildung 7.1: Beispielformular zur Eingabe von Vor- und Nachnamen

Sehen wir uns an, wie man ein einfaches Formular (Abbildung 7.1) erstellt – erst einmal allein den XHTML-Code, noch ohne PHP:

Listing 7.1: Ein Formular mit zwei Textfeldern (formular.php)

```
01 <form action="verarbeitung.php" method="get">
02 Ihr Vorname: <br />
03 <input type="text" name="vorname" size="20" maxlength="30" />
04 <br />
05 Ihr Nachname: <br />
06 <input type="text" name="nachname" size="20" maxlength="30" />
07 <br />
08 <input type="submit" value="Abschicken" />
09 </form>
```

Im Beispielformular werden mehrere (X)HTML-Elemente eingesetzt:

Alle Formularelemente eines Formulars werden innerhalb von `<form>` (Zeile 1) und `</form>` (Zeile 9) geschrieben. Im `form`-Starttag ist daneben noch angegeben, was mit den Formulardaten geschehen soll und wie es geschehen soll:

- Bei `action` steht der Pfad zu einem Skript, das die Verarbeitung übernimmt. Im Beispiel ist es *verarbeitung.php*.
- Bei `method` schreiben Sie die Methode, wie die Formulardaten versendet werden können. Möglich ist neben `get` noch `post`. Genaues zu den Unterschieden in Abschnitt 7.2.

Der Text in Zeile 2 und 5 ist der sichtbare Text, der dem Benutzer sagt, welche Informationen er in den Formularfeldern angeben soll.

In Zeile 3 wird ein Textfeld erzeugt. Hierzu dient das `input`-Element mit dem Attribut `type="text"`. Wichtig ist außerdem, dem Formularfeld einen Namen (`name`) zu vergeben. Über diesen können Sie dann per PHP auf den eingetragenen Inhalt des Formularfelds zugreifen. `size` legt die sichtbare Größe fest und `maxlength` bestimmt, wie viele Zeichen eingegeben werden können.

Achtung

Wenn Sie aber sichergehen möchten, dass wirklich nur eine bestimmte Anzahl an Zeichen eingegeben wird, müssen Sie das noch per PHP prüfen. Mehr hierzu in Abschnitt 7.6.2.

Ein zweites Textfeld für den Nachnamen wird in Zeile 6 erzeugt. Zeile 8 erstellt den Absendebutton über ein `input`-Element mit `type="submit"`. Der bei `value` angegebene Wert wird als Beschriftung des Buttons im Browser angezeigt.

Tipp

Um Ihr Formular benutzerfreundlicher zu machen, sollten Sie zusätzlich das `label`-Element einsetzen. Es dient zur Beschriftung von Formularfeldern. Der Einsatz von `label` macht es möglich, durch einen Klick in die Beschriftung das jeweilige Element auszuwählen, Textfelder erhalten beispielsweise den Fokus.

```
<label for="vorname">Vorname: </label> <input type="text"
name="vorname" size="20" maxlength="40" id="vorname" />
```

Die Verknüpfung von `label` zum dazugehörigen Kontrollelement erreichen Sie über das Attribut `for` bei `label`, das denselben Wert erhält, der über `id` an das Kontrollelement vergeben wird. Dies hat aber auf die Verarbeitung per PHP keine Auswirkung, und deswegen kommen die weiteren Formularbeispiele in diesem Kapitel um der Einfachheit willen ohne `label` aus.

Sehen wir uns jetzt an, wie man per PHP auf die Inhalte zugreifen kann: PHP stellt Ihnen alle Formulardaten in zwei assoziativen Arrays zur Verfügung: `$_POST` für per POST und `$_GET` für per GET versendete Formulardaten. Um auf den Inhalt eines bestimmten Formularfelds zuzugreifen, geben Sie den Namen des Formularfelds als Schlüssel an. Im Beispielformular wird GET als Übertragungsmethode verwendet, deswegen können Sie über `$_GET["vorname"]` und mit `$_GET["nachname"]` auf die eingetragenen Werte der beiden Formularfelder zugreifen.

Tipp

`$_POST` und `$_GET` sind superglobal, das heißt, Sie können auf diese auch innerhalb von Funktionen zugreifen, ohne `global` zu benutzen.

Damit kann die Datei *verarbeitung.php* erstellt werden. Sie soll nur den Inhalt der Formularfelder ausgeben.

```
echo "Ihre Eingaben<br />\n";
echo "Vorname: {$_GET['vorname']}<br />\n";
echo "Name: {$_GET['nachname']}<br />\n";
```

Noch eine Ergänzung: Um *Cross Site Scripting* zu verhindern, müssen Sie alle Inhalte von Formularfeldern *bei der Ausgabe* mit `htmlspecialchars()` behandeln:

Abbildung 7.2: Die eingetragenen Daten werden ausgegeben.

Listing 7.2: Die Ausgabe der Formulardaten (verarbeitung.php)

```
echo "Ihre Eingaben<br />\n";
echo "Vorname: " . htmlspecialchars($_GET["vorname"]) . "<br />\n";
echo "Name: " . htmlspecialchars($_GET["nachname"]) . "<br />\n";
```

Was *Cross Site Scripting* ist und warum Sie es auf die angegebene Art verhindern können, erfahren Sie genauer in Abschnitt 7.6.1.

7.1.1 Verarbeitung im selben Skript

Im Beispiel steht der Code zur Verarbeitung in einer eigenen Datei, häufig werden bei PHP Formular und Verarbeitung im selben Skript untergebracht. Dafür geben Sie bei `action` im `form`-Starttag den Namen des aktuellen Skripts an. Bevor die Ausgabe erfolgt, soll aber überprüft werden, ob das Formular überhaupt abgesendet wurde. Wenn es abgesendet wurde, sind die entsprechenden Variablen gesetzt. Deswegen kann man hier zur Überprüfung zu `isset()` greifen.

Listing 7.3: PHP-Verarbeitung der Formulardaten im selben Skript (formular_ausgabe.php)

```
01 <form action="formular_ausgabe.php" method="get">
02 Ihr Vorname: <br />
03 <input type="text" name="vorname" size="20" maxlength="30" />
04 <br />
05 Ihr Nachname: <br />
06 <input type="text" name="nachname" size="20" maxlength="30" />
07 <br />
08 <input type="submit" value="Abschicken" />
09 </form>
10 <?php
11 if (isset($_GET["vorname"])) {
12   echo "Ihre Eingaben<br />\n"
13     . "Vorname: " . htmlspecialchars($_GET["vorname"])
```

```
14      . "<br />\n Name: " . htmlspecialchars($_GET["nachname"])
15      . "<br />\n";
16 }
17 ?>
```

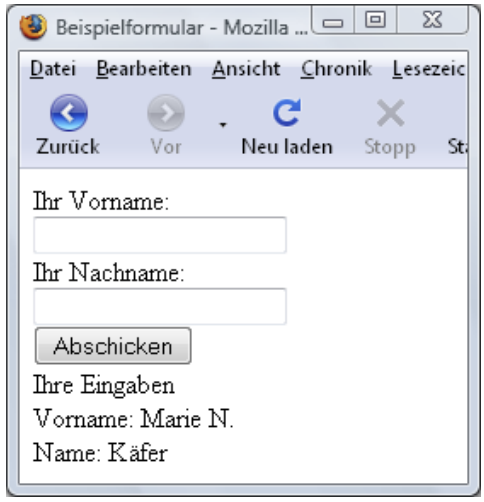


Abbildung 7.3: Unterhalb des Formulars werden die eingegebenen Daten wieder angezeigt.

Im Beispiel soll beim Absenden das Skript selbst aufgerufen werden. Deswegen ist der Name der aktuellen Datei bei `action` eingetragen (Zeile 1).

Das ist aber unpraktisch, denn wenn Sie das Skript unter einem neuen Namen abspeichern, müssen Sie jeweils den Pfad anpassen. Besser ist es, von PHP selbst den Namen des Skripts ermitteln zu lassen, der in der Variablen `$_SERVER["PHP_SELF"]` steht. Damit lässt sich das `form`-Starttag folgendermaßen schreiben:

```
<form action="<?php echo $_SERVER["PHP_SELF"];?>" method="get">
```

Nach `action="` wechseln Sie in den PHP-Modus und lassen die Variable `$_SERVER["PHP_SELF"]` per `echo` ausgeben. Ein Strichpunkt schließt die Anweisung ab, danach wird der PHP-Code-Teil per `?>` beendet und das letzte `"` ist das schließende Anführungszeichen von `action`.

Außerdem sollten Sie auch hier `htmlspecialchars()` benutzen:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="get">
```

Testen Sie damit einmal Ihr Skript. Wenn Sie in den Quellcode wechseln, sehen Sie, dass PHP Ihnen den richtigen Pfad bei `action` automatisch eingetragen hat.

Formular oder Auswertung anzeigen

Wenn Sie nicht wollen, dass bei der Auswertung das Formular selbst erneut angezeigt wird, so lässt sich dies natürlich verhindern:

Listing 7.4: Dieses Mal wird das Formular bei der Auswertung nicht mehr angezeigt (formular_oder_auswertung.php).

```
01 <?php
02 if (!isset($_GET["vorname"])) {
03 ?>
04 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="get">
05 Ihr Vorname: <br />
06 <input type="text" name="vorname" size="20" maxlength="30" />
07 <br />
08 Ihr Nachname: <br />
09 <input type="text" name="nachname" size="20" maxlength="30" />
10 <br />
11 <input type="submit" value="Abschicken" />
12 </form>
13 <?php
14 } else {
15     echo "Ihre Eingaben<br />\n"
16     . "Vorname: " . htmlspecialchars($_GET["vorname"])
17     . "<br />\n Name: " . htmlspecialchars($_GET["nachname"])
18     . "<br />\n";
19 }
20 ?>
```

Jetzt wird die Ausgabe des Formulars von der Bedingung abhängig gemacht, dass die Variable `$_GET["vorname"]` **nicht gesetzt ist**, das Formular also nicht abgesendet wurde. Ansonsten (`else` in Zeile 14) wird die Auswertung angezeigt.

Eingegebene Werte wieder eintragen

Manchmal hingegen möchte man auch bei der Auswertung das Formular anzeigen lassen, aber in den Formularfeldern sollen die bereits eingegebenen Werte stehen. So ist das etwa bei der Suche in Suchmaschinen: Hier erscheint nach der Eingabe eines Suchbegriffs das Formular erneut mit dem eingetragenen Suchbegriff und unten den Suchergebnissen.

Dafür braucht man zuerst einmal das Attribut `value`. Der Wert, den Sie hier angeben, wird im Textfeld angezeigt:

```
<input type="text" name="vorname" size="20" maxlength="30" value="Marie N." />
```

Das können wir im Skript anwenden:

Listing 7.5: Die eingegebenen Werte werden wieder in die Textfelder geschrieben (formular_voreingetragen.php).

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <title>Beispielformular</title>
06   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
07 </head>
08 <body>
09 <?php
10 if (isset($_GET["vorname"])) {
11   $vorname = htmlspecialchars($_GET["vorname"]);
12   $nachname = htmlspecialchars($_GET["nachname"]);
13 } else {
14   $vorname = "";
15   $nachname = "";
16 }
17 ?>
18 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="get">
19 Ihr Vorname: <br />
20 <input type="text" name="vorname" size="20" maxlength="30" value="<?php echo
     $vorname; ?>" />
21 <br />
22 Ihr Nachname: <br />
23 <input type="text" name="nachname" size="20" maxlength="30"  value="<?php echo
     $nachname; ?>" />
24 <br />
25 <input type="submit" value="Abschicken" />
26 </form>
27 <?php
28 if (isset($_GET["vorname"])) {
29   echo "Ihre Eingaben<br />\n";
30   echo "Vorname: $vorname, Name: $nachname<br />\n";
31 }
32 ?>
33 </body>
34 </html>
```

In Zeile 10 wird überprüft, ob das Formular abgesendet wurde. In diesem Fall werden die Variablen `$vorname` und `$nachname` auf die eingegebenen Werte gesetzt, sonst auf einen Leerstring. Im Formular selbst stehen bei den Textfeldern die `value`-Attribute. Bei diesen muss in den PHP-Modus gewechselt und die Inhalte der Variablen per `echo` ausgegeben werden.

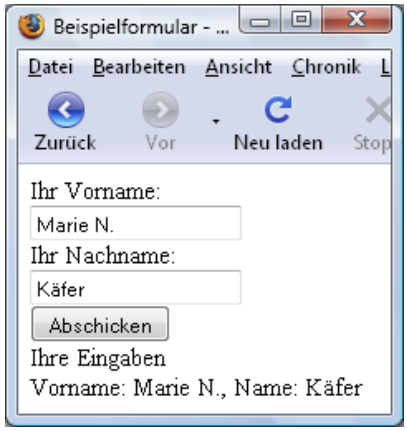


Abbildung 7.4: Die eingegebenen Werte werden im Formular wieder angezeigt.

Übertragung per POST

Bisher wurde GET als Übertragungsmethode eingesetzt. Wenn man das auf POST ändert, müssen auch die PHP-Variablen zum Zugriff auf die Formulareingaben angepasst werden. Sie lauten dann entsprechend `$_POST["vorname"]` und `$_POST["nachname"]`:

Listing 7.6: Die Formulardaten werden dieses Mal per POST übergeben (formular_post.php).

```
01 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="post">
02 <!-- Rest des Formular wie gehabt --></form>
03 <?php
04 if (isset($_POST["vorname"])) {
05   echo "Ihre Eingaben<br />\n"
06     . "Vorname: " . htmlspecialchars($_POST["vorname"])
07     . ", Name: " .htmlspecialchars($_POST["nachname"])
08     . "<br />\n";
09 }
10 ?>
```

Die Ausgabe ist dieselbe wie in Abbildung 7.4.

Tipp

Übrigens stellt Ihnen PHP noch ein weiteres vordefiniertes Array zur Verfügung: `$_REQUEST`. Damit können Sie auf Formulardaten unabhängig davon zugreifen, ob sie per GET oder per POST übertragen wurden. Es beinhaltet außerdem Cookie-Daten (siehe hierzu Kapitel 8).

Mit diesen Skripten als Basis lassen sich jetzt die Unterschiede der zwei Übertragungsmethoden – POST und GET – einmal genauer betrachten.

7.2 Zwei Methoden: POST und GET

Der wesentliche Unterschied bei den Übertragungsmethoden POST und GET ist, dass bei GET die Formulardaten über die URL übertragen werden. Wenn Sie das Skript *formular_ausgabe.php* noch einmal aufrufen, etwas eintragen und dann die Adresszeile im Browser betrachten, sehen Sie es deutlich:

Abbildung 7.5: Bei der Übertragung mit GET wird die URL modifiziert.

Hinter der URL des Skripts folgt ein Fragezeichen und dann der Name des ersten Formularfelds und nach einem = der eingetragene Wert. Im Beispiel ist es `Marie N.` Das Leerzeichen wird durch ein +-Zeichen ersetzt. Nach einem Ampersand-Zeichen `&` folgt das zweite Formularfeld. Wieder ist der Name des Felds mit = vom eingegebenen Wert getrennt. Im Beispiel wurde *Käfer* eingegeben, das *ä* wird hier URL-kodiert wiedergegeben.

Tipp

Dieser Aufbau ist praktisch, da Sie auch selbst in Ihren Skripten auf diese Art Daten an ein anderes Skript übergeben können – Sie basteln die URL einfach entsprechend zusammen und fügen sie in einen Link ein.

```
<a href="beispiel.php?nr=2&kat=56">...</a>
```

Bzw. da Sie in (X)HTML das Ampersandzeichen maskieren müssen:

```
<a href="beispiel.php?nr=2&amp;kat=56">...</a>
```

Da GET die normale Methode ist, in der der Browser Webseiten vom Server anfordert, ist zur Übertragung von Informationen über eine URL nicht mehr notwendig. Für den Webserver oder das verarbeitende Skript ist es auch *nicht ersichtlich*, ob die Daten aus einem Formular stammen oder nicht. Ein Beispiel hierzu sehen Sie in Kapitel 11.

Dass die Daten bei GET per URL übertragen werden, hat mehrere Vorteile und auch Nachteile:

- An sich kann man mit dieser URL alles machen, was man mit URLs so machen kann – ein Benutzer kann sie beispielsweise in die Lesezeichen aufnehmen und die Parameter werden mit aufgenommen. Das macht GET praktisch für Suchanfragen.

Suchmaschinen wie Yahoo! oder Google verwenden beispielsweise GET bei der Verarbeitung der ins Suchformular angegebenen Daten: Das bedeutet, dass man eine Yahoo!-Abfrage auch in die Lesezeichen aufnehmen kann, später aufrufen und das aktuelle Ergebnis ansehen kann.

- Die Daten in der URL sind natürlich sichtbar – zur Versendung von Login-Daten samt Passwort wäre GET eindeutig die falsche Wahl.
- Die URL samt Parameter kann auch in der Browser-History gespeichert werden.
- Die Daten lassen sich direkt vom Surfer editieren. Das verleitet dazu, mal eben rasch auszuprobieren, was passiert, wenn man hier einen Wert ändert.
- Die Menge der Daten, die übertragen werden können, ist begrenzt.

Bei der Entwicklung und bei ersten Tests hat GET einige Vorteile – durch die Sichtbarkeit der übertragenen Daten können Sie kontrollieren, was und wie es übertragen wird. Wenn Sie allerdings auf den Reload-Button klicken, um das Skript neu zu laden, werden die vorher eventuell bereits eingetragenen Daten erneut gesendet. Bei Bedarf können Sie diese aus Ihrer URL löschen.

Bei der Datenübertragung per POST werden die Formulardaten nicht über die URL, sondern im Dokumentkörper mit übertragen. Sie sind auf normalem Wege nicht sichtbar.

Tipp

Es gibt aber verschiedene Tools, um diese sichtbar zu machen – beispielsweise die Firefox-Erweiterung Live HTTP Headers (*https://addons.mozilla.org/de/firefox/addon/3829*).

Prinzipiell gilt: Immer wenn viele Daten übertragen werden oder auch sensible Daten, oder wenn das Formular verwendet wird, um Änderungen in einer Datenbank zu speichern, sollten Sie unbedingt POST verwenden, ansonsten GET.

7.3 Weitere Formularelemente

Bisher wurden nur die Textfelder und der Absende-Button besprochen. Sehen wir uns jetzt weitere Formularelemente an.

7.3.1 Radiobuttons, Auswahllisten und mehrzeilige Textfelder

Radiobuttons, Auswahllisten und mehrzeilige Textfelder bieten, was den Zugriff auf die eingegebenen Daten per PHP anbelangt, wenige Besonderheiten und werden deswegen gemeinsam besprochen. Zuerst einmal das Beispielformular, in dem all diese Formularfelder vereint sind.

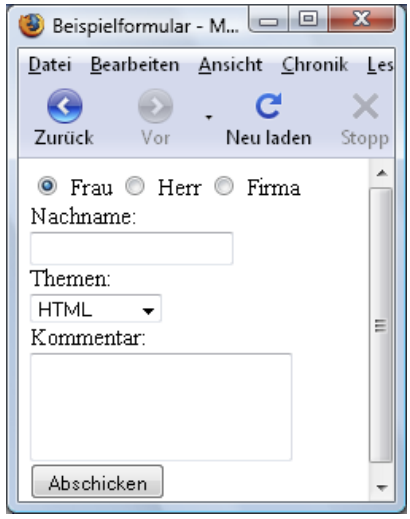


Abbildung 7.6: Radiobuttons, Auswahlliste und mehrzeiliges Textfeld

```
01 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="get">
02 <input type="radio" name="anrede" value="Frau" checked="checked"/> Frau
03 <input type="radio" name="anrede" value="Herr" /> Herr
04 <input type="radio" name="anrede" value="Firma" /> Firma <br />
05 Nachname: <br />
06 <input type="text" name="nachname" size="20" maxlength="30" />
07 <br />
08 Themen: <br />
09 <select name="thema">
10   <option value="HTML">HTML</option>
11   <option value="CSS">CSS</option>
12   <option value="JavaScript">JavaScript</option>
13   <option value="PHP">PHP</option>
14 </select>
15 <br />
16 Kommentar: <br />
17 <textarea name="kommentar" rows="3" cols="20"></textarea>
18 <br />
19 <input type="submit" value="Abschicken" />
20 </form>
```

Im Beispiel wird die Anrede über Radiobuttons realisiert. Von einer Gruppe von Radiobuttons kann ein Benutzer immer nur eines auswählen. Damit das funktioniert, müssen die zusammengehörigen Radiobuttons denselben Namen erhalten (im Beispiel `anrede`). Was hingegen übertragen wird, steht beim Attribut `value`. Soll ein Radiobutton zu Anfang schon aktiviert sein, notieren Sie zusätzlich `checked="checked"` (Zeile 2).

In den Zeilen 9–14 wird eine Auswahlliste erstellt. Auswahllisten werden über `select` und `option` realisiert. `select` ist das umfassende Element, dem Sie einen Namen geben müssen. Die einzelnen Auswahlpunkte werden in `option`-Elementen geschrieben. Standardmäßig ist der erste Punkt der Liste vorausgewählt. Soll es ein anderer sein, so ergänzen Sie beim gewünschten Punkt `selected="selected"`.

In Zeile 17 sehen Sie ein mehrzeiliges Textfeld. Das mehrzeilige Textfeld wird über ein eigenes Element mit Namen `textarea` realisiert. Über `rows` und `cols` legen Sie die Größe fest: `rows` bestimmt die Anzahl an Zeilen, `cols` die Anzahl an Zeichen nebeneinander. Wenn jemand mehr eingibt, erscheinen automatisch Scrollleisten. Soll schon Text im mehrzeiligen Textfeld stehen, so wird dieser zwischen `<textarea>` und `</textarea>` notiert.

Nun zur Auswertung per PHP, die direkt darunter erfolgt:

Listing 7.7: Mit PHP auf weitere Formularfelder zugreifen (formular_weitere.php)

```
21 <?php
22 if (!empty($_GET["nachname"])) {
23   echo "Ihre Eingaben<br />\n";
24   if (!empty($_GET["anrede"])) {
25     echo htmlspecialchars($_GET["anrede"]);
26   }
27   echo " " . htmlspecialchars($_GET["nachname"]) . "<br />\n";
28   if (!empty($_GET["thema"])){
29     echo "Das gewählten Thema: ". htmlspecialchars($_GET["thema"]) . "<br />\n ";
30    }
31   if (!empty($_GET["kommentar"])) {
32     echo "Ihr Kommentar: " . htmlspecialchars($_GET["kommentar"]);
33   }
34 }
35 ?>
```

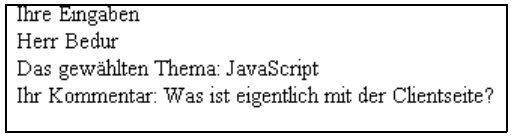


Abbildung 7.7: Eine Beispielausgabe von eingegebenen Daten

Dieses Mal wird nicht `isset()` zur Überprüfung verwendet, sondern `!empty()`. Damit prüfen Sie gleichzeitig, ob *eine Variable gesetzt und nicht leer* ist. Nur in diesem Fall werden die Inhalte ausgegeben. Auf diese greifen Sie zu wie bei den Textfeldern.

Eine Ergänzung zur Ausgabe des Inhalts des `textarea`-Elements: Falls jemand einen Zeilenumbruch in einem mehrzeiligen Textfeld macht, wird dieser natürlich nicht wieder ausgegeben. Wenn Sie das jedoch möchten, können Sie die Funktion `nl2br()` einsetzen. Diese verwandelt `\n` in `<br />`:

*Listing 7.8: Ins Textfeld eingegebene Zeilenumbrüche werden in
 umgewandelt und damit wieder angezeigt (formular_textarea_nl.php).*

```
if (!empty($_GET["kommentar"])){
    echo "Ihr Kommentar: ";
    echo nl2br(htmlspecialchars($_GET['kommentar']));
    echo "<br />\n";
}
```

Achtung

Zuerst müssen Sie `htmlspecialchars()` auf den Inhalt anwenden und dann `nl2br()`: Sonst wandeln Sie die gerade erzeugten `<br />`-Zeichen in `<br />` um und das ist nicht erwünscht.

`nl2br()` erzeugt standardmäßig Zeilenumbrüche im XHTML-Format, d. h. `<br />`. Wenn Sie hingegen HTML verwenden, können Sie als zweiten Parameter `false` übergeben, dann werden die Zeilenumbrüche HTML-kompatibel ausgegeben, also `<br>`. Diese Option steht seit PHP 5.3 zur Verfügung:

```
echo nl2br("Nicht \n ganz \n ungebrochen", false);
```

7.3.2 Checkboxen

Fehlen noch die Checkboxen. Im Unterschied zu den Radiobuttons kann man bei einer Gruppe von Checkboxen mehrere ankreuzen.

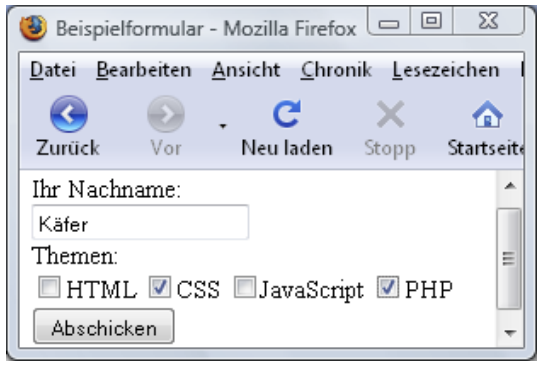


Abbildung 7.8: Bei Checkboxen kann man mehrere gleichzeitig ankreuzen.

Checkboxen sind ebenfalls `input`-Elemente, allerdings dieses Mal vom `type="checkbox"`. Wie alle Formularelemente brauchen sie einen Namen. Hier gibt's aber etwas zu beachten. Mehrere Checkboxen können gleichzeitig angekreuzt werden, das heißt, es sollen mehrere Werte übertragen werden. Mehrere Werte über einen Namen ansprechen – genau dafür sind Arrays da. Damit PHP die Werte aller angekreuzten Checkboxen als Array abspeichert, müssen Sie beim Namen *eckige Klammern* schreiben.

```
<input type="checkbox"  name="thema[]" value="PHP" />
```

Tipp

Wenn Sie die eckigen Klammern weglassen, würde nur der letzte Wert abgespeichert werden.

Damit sieht das Beispielformular folgendermaßen aus:

```
01 <form action="<?php echo $_SERVER["PHP_SELF"];?>" method="get">
02 Ihr Nachname: <br />
03 <input type="text" name="nachname" size="20" maxlength="30" />
04 <br />
05 Themen: <br />
06 <input type="checkbox" name="thema[]" value="HTML" />HTML
07 <input type="checkbox" name="thema[]" value="CSS" />CSS
08 <input type="checkbox" name="thema[]" value="JavaScript" />JavaScript
09 <input type="checkbox" name="thema[]" value="PHP" />PHP<br />
10 <input type="submit" value="Abschicken" />
11 </form>
```

Es folgt wieder die Auswertung per PHP:

Listing 7.9: Checkboxen (formular_checkbox.php)

```
12 if (!empty($_GET["nachname"])) {
13   echo "Ihre Eingaben<br />\n"
14     . "Name: ". htmlspecialchars($_GET["nachname"]) . "<br />\n";
15   if (isset($_GET["thema"]) && is_array($_GET["thema"])){
16     echo "Die gewählten Themen: <br />\n ";
17     foreach($_GET["thema"] as $th) {
18       echo htmlspecialchars($th) . "<br />\n";
19     }
20   }
21 }
```

Durch die Überprüfung in Zeile 12 wird sichergestellt, dass die Ausgabe nur erfolgt, wenn das Formular abgesendet wurde und bei Nachname etwas eingetragen wurde.

Die Ausgabe der Themen beginnt in Zeile 16. Zuerst wird überprüft, ob die Variable gesetzt ist. Denn wenn keine Checkbox angekreuzt ist, wird auch nichts übertragen. Außerdem stellt `is_array()` sicher, dass `$_GET["thema"]` wirklich ein Array ist. Nur dann wird in einer `foreach`-Schleife der Inhalt ausgegeben.

Hinweis

`$_GET` ist ja das von PHP bereitgestellte assoziative Array. In diesem Fall enthält es mehrere Werte und auch eben als einen Wert wiederum das Array für die Werte der Checkbox, es ist damit im Beispiel ein verschachteltes Array.

Eine Ausgabe mit `print_r()` zeigt den Aufbau:

```
Array
(
    [nachname] => Käfer
    [thema] => Array
        (
            [0] => CSS
            [1] => JavaScript
        )

)
```

Abbildung 7.9: Der Aufbau des verschachtelten $_GET-Arrays im Beispiel

Im Beispiel wurden die gewählten Themen in Zeile 18 mit einem Zeilenumbruch getrennt ausgegeben. Je nachdem, wie diese Daten weiterverarbeitet werden sollen, brauchen Sie diese Daten aber auch in einem String. Dafür können Sie `implode()` einsetzen.

Modifizieren wir die Ausgabe der gewählten Themen entsprechend:

Listing 7.10: Die Themen als String (formular_checkbox_implode.php)

```
if (isset($_GET["thema"]) && is_array($_GET["thema"])){
  echo "Die gewählten Themen: <br />\n ";
  $themen = implode(" ", $_GET["thema"]);
  echo htmlspecialchars($themen);
}
```

Die Themen werden jetzt durch Leerzeichen getrennt aneinander gehängt in der Variable `$themen` gespeichert. So könnte man sie auch in einer Datenbank speichern oder per Mail versenden.

Hinweis

Ein weiteres praktisches Formularfeld ist das versteckte Feld:

```
<input type="hidden" name="id" value="123" />
```

Es ist vom Typ `hidden` und wird in der Browserausgabe nicht angezeigt. Durch das obige Feld wird beim Absenden des Formulars `id=123` übertragen. Das versteckte Feld dient dazu, Daten zwischen verschiedenen Seiten zu übertragen und verhält sich wie ein normales Formularfeld mit dem einzigen Unterschied, dass es in der normalen Browseransicht nicht angezeigt wird. Ein Beispiel zur Verwendung sehen Sie in Kapitel 11.

7.4 In PHP 5.3 zu Recht deprecated: die Magic Quotes

Ein Punkt wurde noch nicht erwähnt, die *Magic Quotes*. Wenn diese Einstellung aktiviert ist, wird bei allen Daten aus Formularen (GET- und POST-Daten) und aus Cookies (siehe Kapitel 8) automatisch ein Backslash vor `'`, `"`, `\` und vor Nullzeichen ergänzt.

Nehmen wir an, jemand gibt in das Beispielformular *formular_weitere.php* in das mehrzeilige Textfeld ein *Hallo, wie geht's? Hast du »Himmel und Hölle« inzwischen gelesen?* Dann sieht die Eingabe bei aktiviertem Magic Quotes, wenn sie wieder ausgegeben wird, wie in Abbildung 7.10 aus.

```
Ihr Kommentar: Hallo, wie geht\'s? Hast du \"Himmel und Hölle\" inzwischen gelesen?
```

Abbildung 7.10: Magische \-Zeichen erscheinen vor ' und ".

Magic Quotes sollen die SQL-Injection – ein großes Sicherheitsproblem bei der Arbeit mit Datenbanken (siehe Kapitel 11) – erschweren. Magic Quotes sind in zweierlei Hinsicht problematisch: Zuerst einmal schützen sie nicht so zuverlässig wie die speziell zur Maskierung von Daten vorgesehenen Funktionen des jeweiligen Datenbankmanagementsystems und zum anderen sind die Backslashes bei der Ausgabe per (X)HTML oder beim Versenden von Formulardaten per E-Mail unerwünscht. Deswegen ist es besser, `magic_quotes_gpc` in der *php.ini* abzuschalten und geeignete Vorkehrungen zur Verhinderung von SQL-Injections selbst zu treffen (mehr dazu in Kapitel 11).

Dies ist auch das empfohlene Vorgehen: Seit PHP 5.3 erhalten Sie sogar eine Deprecated-Meldung, wenn Sie Magic Quotes aktiviert haben.

```
PHP Warning:  Directive 'magic_quotes_gpc' is deprecated in PHP 5.3 and greater in
PHP Warning:  Directive 'magic_quotes_runtime' is deprecated in PHP 5.3 and greater
PHP Warning:  Directive 'magic_quotes_sybase' is deprecated in PHP 5.3 and greater
```

Abbildung 7.11: PHP-Warnung in den Fehlerlogs von Apache bei aktivierten Magic Quotes

Und da es Magic Quotes in PHP 6 nicht mehr geben soll, ist dieser Weg auch zukunftskompatibel.

Deaktiviert werden Magic Quotes durch folgende Einstellungen in der *php.ini*-Datei:

```
magic_quotes_gpc = Off;
magic_quotes_runtime = Off
magic_quotes_sybase = Off
```

Wenn Sie keinen Zugriff auf die *php.ini*-Datei haben, können Sie diese Konfiguration auch über eine *.htaccess*-Datei (mehr hierzu im Anhang) erledigen, die Sie im Verzeichnis mit Ihren PHP-Skripten abspeichern. Hier schreiben Sie folgende Zeile:

```
php_flag magic_quotes_gpc Off
```

Soll Ihr Skript unabhängig von der aktuellen PHP-Konfiguration funktionieren, können Sie über `get_magic_quotes_gpc()` ermitteln, ob Magic Quotes aktiviert sind oder nicht. Diese Funktion gibt bei aktivierten Magic Quotes 1 zurück, ansonsten 0. Außerdem brauchen Sie noch die Funktion `stripslashes()` zur Entfernung der Backslashes bei der Ausgabe der einzelnen Formularfelder.

Das folgende Beispiel zeigt eine Funktion zum Entfernen der Backslashes nach Bedarf – d. h. nur bei aktivierten Magic Quotes.

Listing 7.11: Jetzt wird stripslashes() nur bei Bedarf angewandt (formular_mq_entfernen.php).

```
01 function mqentf($var)
02 {
03   if (get_magic_quotes_gpc()) {
04     return stripslashes($var);
05   }  else {
06     return $var;
07   }
08 }
09 if (!empty($_GET["kommentar"])) {
10     echo "Ihr Kommentar: "
11     . htmlspecialchars(mqentf($_GET["kommentar"]));
12 }
```

Problematisch ist am gerade vorgestellten Skript: `stripslashes()` erwartet einen String, bei einem Array gäbe es eine unschöne Fehlermeldung. Eine globale Lösung – das heißt, die Backslashes werden bei allen `$_GET`-, `$_POST`- und Cookie-Werten entfernt, zeigt das PHP-Manual (*http://de.php.net/manual/en/security.magicquotes.disabling.php*):

```
01 if (get_magic_quotes_gpc()) {
02   function stripslashes_deep($value)
03   {
04     $value = is_array($value) ?
```

```
05              array_map('stripslashes_deep', $value) :
06              stripslashes($value);
07     return $value;
08   }
10   $_POST = array_map('stripslashes_deep', $_POST);
11   $_GET = array_map('stripslashes_deep', $_GET);
12   $_COOKIE = array_map('stripslashes_deep', $_COOKIE);
13   $_REQUEST = array_map('stripslashes_deep', $_REQUEST);
15 }
```

Zeile 1 überprüft, ob die Bearbeitung notwendig ist. Wenn Magic Quotes aktiviert sind, gibt `get_magic_quotes_gpc()` 1 zurück und der angegebene Code wird ausgeführt. In Zeile 2 beginnt die Funktionsdefinition für `stripslashes_deep()`. Der Name deutet schon darauf hin: Diese Funktion soll auch in die Tiefen verschachtelter Arrays wirken. In Zeile 4 wird überprüft, ob es sich bei diesem der Funktion übergebenen Parameter um ein Array handelt: Ist dies der Fall, wird `array_map()` aufgerufen. Darüber kann man eine Funktion auf alle Elemente des Arrays anwenden. `array_map()` erwartet zwei Parameter: zuerst den Namen der Funktion, die aufgerufen werden soll, und als zweiten Parameter das Array. Im Beispiel wird `stripslashes_deep()` selbst wieder aufgerufen. Handelt es sich hingegen nicht um ein Array, wird `stripslashes()` direkt bei `$value` ausgeführt. Schließlich wird in Zeile 7 der bereinigte Wert zurückgegeben. In den Zeilen 10 bis 13 wird `stripslashes_deep()` auf alle Arrays angewendet, bei denen die Magic Quotes tätig werden: auf `$_POST`- und `$_GET`-Daten und außerdem auf `$_COOKIE` (mehr hierzu in Kapitel 8) und `$_REQUEST`.

Die Funktion ist natürlich schwerfällig. Eindeutig besser ist es, Magic Quotes abzuschalten. Und: Gut, dass diese problematische Einstellung in PHP 6 gestrichen sein wird.

7.5 Sicherheit – misstrauen Sie Ihren Besuchern

Bei Formularen gibt es einiges in puncto Sicherheit zu beachten.

7.5.1 Gefährliche Einstellung: register_globals = On

Sie haben bisher gesehen, dass man über die superglobalen Arrays `$_GET` und `$_POST` auf in Formulare eingetragene Werte zugreifen kann. Sofern die *php.ini*-Einstellung `register_globals` auf `On` gestellt ist, geht das jedoch auch wesentlich einfacher: Dann können Sie beispielsweise anstelle von `$_GET["vorname"]` auch direkt über `$vorname` auf den Inhalt des Felds mit dem Namen `vorname` zugreifen. Diese Option wird allerdings in PHP 6 nicht mehr zur Verfügung stehen und gibt ab PHP 5.3 eine deprecated-Meldung aus.

Tipp

Die superglobalen Arrays `$_POST` und `$_GET` funktionieren bei aktivierten `register_globals` aber trotzdem.

Auf den ersten Blick erscheint der Zugriff über die einfachen Variablen praktisch. In Kombination mit etwas schlampigem Programmierstil ist diese Einstellung allerdings sehr gefährlich, wie das folgende Beispiel zeigt:

Listing 7.12: Bei aktivierten register_globals enthält dieses Skript eine Sicherheitslücke (register_globals.php).

```
01 if (false) {
02   $superuser = true;
03 }
04 if (isset($superuser) && $superuser == true) {
05   echo "Sie dürfen rein";
06 } else {
07   echo "Sie dürfen nicht rein";
08 }
```

Hier wird in Zeile 1 eine Überprüfung durchgeführt. Im Beispiel steht als Bedingung nur `false`, d. h. die Bedingung ist nicht wahr. Sie können sich hier aber auch vorstellen, dass eine komplexere echte Überprüfung stattfindet. Falls diese klappt, wird der Variablen `$superuser` der Wert `true` zugewiesen.

In Zeile 4 wird überprüft, ob die Variable `$superuser` gesetzt ist und den Wert `true` hat. In diesem Fall wird »Sie dürfen rein« ausgegeben, ansonsten »Sie dürfen nicht rein«. So wie das Beispiel konstruiert ist, ist eigentlich klar, dass nur »Sie dürfen nicht rein« ausgegeben werden kann. Oder?

Anders sieht es aus, wenn jemand *bei aktivierten* `register_globals` das Skript folgendermaßen aufruft:

```
http://localhost/php-beispiele/register_globals.php?superuser=true
```

Dann wird nämlich eine neue Variable `$superuser` kreiert und auf den Wert `true` gesetzt. Und damit wird ungeachtet von `if(false) { $superuser = true; }` ausgegeben »Sie dürfen rein".

Verhindern lässt sich das natürlich ganz einfach: Man muss nur am Anfang die Variable `$superuser` initialisieren. Die Überprüfung mit `isset()` kann dann in der zweiten `if`-Verzweigung entfallen.

Listing 7.13: Abgesichert durch Initialisierung der Variable (register_globals_sicher.php)

```
01 $superuser = false;
02 if (false) {
03   $superuser = true;
04 }
05 if ($superuser == true) {
06   echo "Sie dürfen rein";
07 } else {
08   echo "Sie dürfen nicht rein";
09 }
```

Ruft man jetzt bei aktivierten `register_globals` das Skript mit `http://localhost/php-beispiele/register_globals.php?superuser=true` auf, wird trotzdem wie gewünscht »Sie dürfen nicht rein« ausgegeben.

Sie sollten sich angewöhnen, Variablen immer zu initialisieren, damit Ihre Skripte keine Sicherheitslücken aufweisen, falls sie bei einem Provider eingesetzt werden, bei dem `register_globals` auf `on` steht.

Tipp

Über eine *.htaccess*-Datei mit der folgenden Zeile können Sie `register_globals` ausschalten:

```
php_flag register_globals off.
```

7.5.2 Bösartige Formulareingaben

Bisher haben wir alle Formularinhalte für die Ausgabe immer mit `htmlspecialchars()` behandelt. Sehen wir uns einmal an, warum das notwendig ist. Häufig wird der Inhalt, der von einem Nutzer in ein Formularfeld eingegeben wurde, wieder auf einer Webseite ausgegeben. Bei einem Blog gibt es beispielsweise üblicherweise die Möglichkeit, einen Kommentar zu hinterlassen, der gespeichert und anderen Benutzern angezeigt wird.

Bilden wir diesen Fall einmal etwas vereinfacht nach. Vereinfacht bedeutet, dass der Inhalt direkt ausgegeben wird – schlimmer noch wird die Sicherheitslücke allerdings, wenn die Inhalte dann auch anderen Benutzern gezeigt werden, da sie z. B. in einer Datenbank gespeichert und von dort wieder ausgelesen werden. Aber an dem gezeigten Beispiel sollte das Grundprinzip gut deutlich werden.

Listing 7.14: Beispielformular für Manipulationen (formular_sicherheit.php)

```
01 <form method="get" action="<?php echo $_SERVER["PHP_SELF"]; ?>">
02 Suchbegriff: <br />
03 <input type="text" name="suche" size="80">
04 <br />
05 <input type="submit"><br />
06 <?php
07 if(isset($_GET["suche"])) {
08   echo "Ihre Suche nach <hr />" . stripslashes($_GET["suche"])
09        . "<hr /> hat folgende Ergebnisse ...";
10 }
11 ?>
12 <p>Lorem ipsum dolor sit amet... amet. <!-- und viel weiterer Text --></p>
```

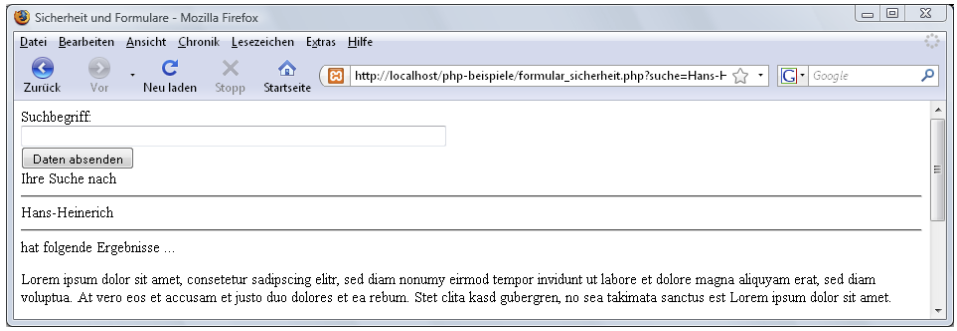


Abbildung 7.12: So sollte es sein ...

Im Listing sehen Sie ein einfaches Formular mit einem Formularfeld (Zeile 11). Der Inhalt dieses Formularfelds wird in Zeile 16 ausgegeben – ohne `htmlspecialchars()`. Darunter stehen einige Absätze Blindtext. Abbildung 7.12 zeigt, wie es sein sollte: Ein Suchbegriff wird eingegeben und auf der Webseite wieder ausgegeben.

Je nachdem, was ein Benutzer eingibt, können aber ganz andere Dinge geschehen. Am besten testen Sie das selbst einmal mit der angegebenen Datei.

- Gibt jemand in das Formularfeld `<div style=background-color:red>` ein, so wird der ganze untere Bereich rot eingefärbt.
- Auch andere optische Entstellungen sind möglich: Durch die Eingabe von `<div style="background: url(http://www.google.de/images/nav_logo3.png)">` wird die Webseite mit einem Hintergrundbild versehen.
- Auch die Eingabe von `<style>` hat eine relativ große Auswirkung: Der ganze untere Text verschwindet.
- Aber Benutzer können bei dieser Datei auch JavaScript-Code ausführen lassen: etwa durch die Eingabe von `<script> alert(42)</script>` in das Formularfeld. In diesem Fall wird die Zahl 42 in einem `alert`-Fenster ausgegeben.

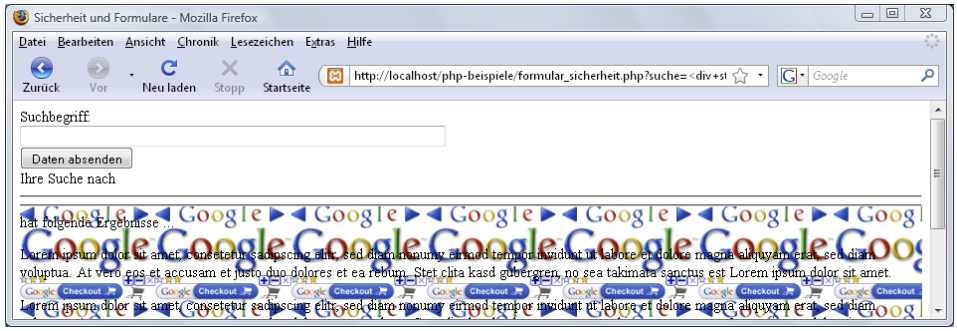


Abbildung 7.13: Auch das war wahrscheinlich nicht geplant ... durch die Eingabe wird ein Hintergrundbild eingebunden.

- Der Besucher könnte aber auch anderen bösartigeren JavaScript-Code ausführen. Durch die Eingabe des folgenden Codes geschieht eine automatische Umleitung auf *http://www.example.com/*.

  ```
  <script>window.location.href="http://www.example.com/"</script>
  ```

  Hier kann eine beliebige Adresse eingegeben werden und damit eine automatische Umleitung auf eine Seite stattfinden, die z. B. versucht, einen Trojaner zu installieren. Und bei dem letzten Beispiel wie auch bei allen vorher aufgeführten Beispielen müssen Sie sich vorstellen, dass bei einer Kommentar-Funktion o. Ä. der schadhafte Beitrag ja allen Besuchern angezeigt wird, die danach auf die Webseite kommen.

Hinweis

Zugegebenermaßen funktioniert das letzte Beispiel in dieser einfachen Form nur, wenn `stripslashes()` angewendet wird. Aber es gibt komplizierter anzuwendende Techniken, wie man eine Umleitung auch realisieren kann, wenn Anführungszeichen mit Backslash escaped werden. Viele Tricks hierzu – und damit viele Tricks, was man bei der Sicherheit berücksichtigen muss, finden sich unter *http://ha.ckers.org/xss.html*.

In dem Moment, in dem ein Besucher ungestört JavaScript-Code in ein Formular eingeben kann und dieser ausgeführt wird, ist noch wesentlich mehr möglich:

- Login-Daten von Benutzern, die später auf die befallene Seite surfen, können ausgelesen und an einen fremden Server versendet werden.
- Auch der Inhalt von Cookies kann an einen fremden Server geschickt werden.

Das Ausführen von bösartigem JavaScript-Code wird als *Cross Site Scripting* (XSS) bezeichnet.

Das Problem, mit dem wir es hier zu tun haben, ist, dass Benutzer in das Formular *selbst (X)HTML-Code eingeben* und dieser genauso behandelt wird wie der andere

(X)HTML-Code der Seite auch. Selbstverständlich gibt es Möglichkeiten, das zu unterbinden. Die einfachste ist, die Sonderzeichen, die es in (X)HTML gibt, das heißt auf jeden Fall < und >, durch die entsprechenden Entities zu ersetzen. Das macht die Funktion `htmlspecialchars()` – und deswegen sollten Sie sie immer bei der Ausgabe von Formularinhalten einsetzen.

Listing 7.15: Minimale Absicherung: Behandlung der Daten mit htmlspecialchars() (formular_htmlspecialchars.php)

```
if(isset($_GET["suche"])) {
  echo "Ihre Suche nach <hr />" . htmlspecialchars(stripslashes($_GET["suche"])) . "
<hr /> hat folgende
Ergebnisse ...";
```

Wenn Sie jetzt im Formular den Beispiel-Schadcode von oben eingeben, werden Sie keinen Erfolg haben: Da die spitzen Klammern in das entsprechende (X)HTML-Entity `<` und `>` umgewandelt werden, wird dieser Code zwar angezeigt, aber eben nicht mehr als (X)HTML-Quellcode ausgeführt und das heißt, er ist nicht mehr gefährlich.

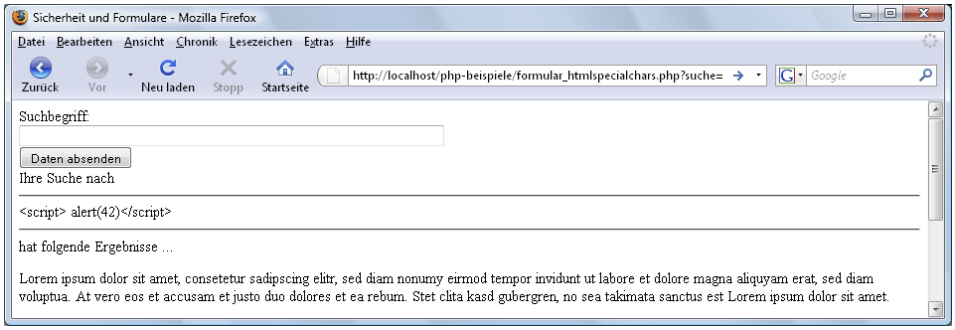


Abbildung 7.14: Jetzt wird der JavaScript-Code nicht ausgeführt und auch die anderen Manipulationen funktionieren nicht mehr.

Welche Eingaben müssen mit `htmlspecialchars()` behandelt werden? Auf den ersten Blick könnte man vermuten, dass die Anwendung von `htmlspecialchars()` auf Textfelder und Textarea-Felder reichen könnte. Bei den anderen kann man doch nichts anderes eingeben? Oder?

7.5.3 Formulare manipulieren

Erst einmal können selbstverständlich alle per GET versendeten Formulardaten direkt manipuliert werden. So kann man beim Listing (Listing 7.7) die URL direkt editieren. Aus der ursprünglichen Version

```
http://localhost/php-beispiele/
formular_weitere.php?anrede=Frau&nachname=Muser&thema=JavaScript
```

lässt sich natürlich leicht eine nicht vorgesehene Anrede machen oder auch schadhafter Code integrieren:

```
http://localhost/php-beispiele/formular_weitere.php?anrede=<script>alert(42)</script>
```

Aber – und dieser Punkt ist entscheidend – das geht *prinzipiell auch bei per POST-versendeten Formulardaten*. Beispielsweise ganz komfortabel über die Firefox-Erweiterung Web Developer Toolbar (*http://www.erweiterungen.de/detail/Web_Developer/*): Diese hat einen eigenen Menüpunkt FORMULARE.

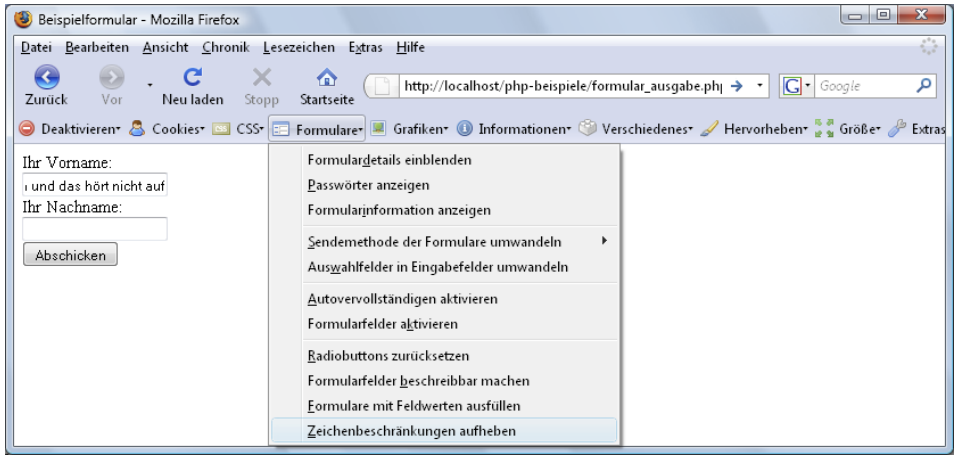


Abbildung 7.15: Viele interessante Optionen für Formulare bei der Web Developer Toolbar

Die ersten drei Punkte bieten zusätzliche Informationen über Formulare. Darunter haben Sie unter anderem folgende Optionen:

- AUSWAHLFELDER IN EINGABEFELDER UMWANDELN: Diese Option macht genau das, was ihr Name andeutet: Aus über das (X)HTML-Element `select` erstellten Auswahllisten werden normale Eingabefelder. Das heißt: Auch wenn Sie bei einer `select`-Auswahlliste nur bestimmte Optionen vorgeben, kann ein Benutzer hier beliebige Inhalte eingeben und damit natürlich auch schadhaften Code.
- RADIOBUTTONS ZURÜCKSETZEN: Haben Sie einen Radiobutton aktiviert, können Sie ihn nur deaktivieren, wenn Sie einen anderen Radiobutton anklicken. Mit dieser Option können hingegen alle Radiobuttons abgewählt werden.
- ZEICHENBESCHRÄNKUNGEN AUFHEBEN: Die bei Textfeldern über `maxlength` angegebene Höchstzeichenzahl wird hier aufgehoben.

Aber natürlich sind das nicht alle Möglichkeiten: Die volle Freiheit hat ein Surfer, der den HTML-Quellcode eines Formulars aus dem Internet lokal bei sich auf der Fest-

platte abspeichert. Dann kann er alle gewünschten Änderungen durchführen. Danach muss er nur bei `action` im `form`-Starttag den absoluten Pfad zum verarbeitenden Skript im Internet angeben. Dann kann er das Formular lokal aufrufen und absenden.

Wenn Sie es bei einem Ihrer Skripte einmal austesten wollen, müssen Sie beispielsweise bei `action` den Pfad mit `http://localhost/php-beispiele/name-des-verarbeitenden-skripts.php` angeben und das Formular wird an das verarbeitende Skript gesendet.

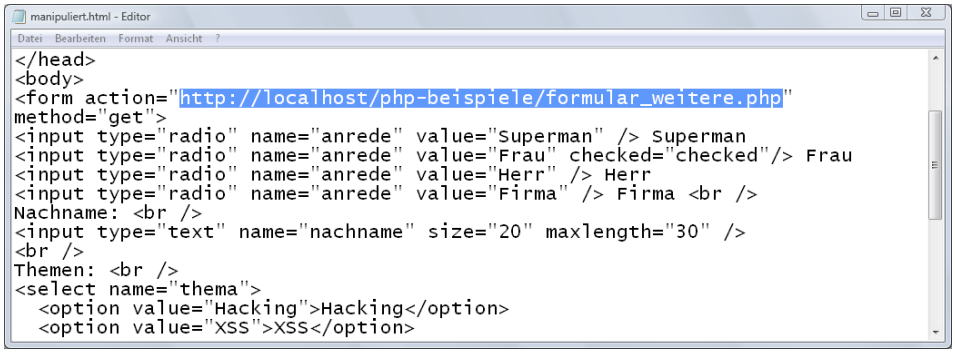


Abbildung 7.16: Jedes beliebige Formular kann man lokal abspeichern, verändern (Superman als zusätzliche Anrede im Beispiel, XSS und Hacking als Thema) und dann das veränderte Formular absenden.

Diese Beispiele sollten deutlich gezeigt haben, dass die *Gefahr von XSS bei allen über Formulare erhaltenen Daten* droht – ganz unabhängig davon, ob es Checkboxen, Auswahllisten oder Textfelder sind.

Neben den `$_POST`, `$_GET` und `$_COOKIE`-Daten, zu denen Sie mehr im Kapitel 8 erfahren, betrifft das ebenfalls HTTP-Header. Was genau HTTP-Header sind, erfahren Sie ebenfalls in Kapitel 8.

7.6 Formulare absichern

Es gibt zwei wichtige Strategien, die Sie bei Formulardaten anwenden sollten:

- Output maskieren, um XSS zu verhindern
- Input filtern – damit keine unerwarteten Dinge in Ihrer Datenbank o. Ä. gespeichert werden

Zu diesen beiden Punkten gehen wir jetzt ins Detail.

7.6.1 Output maskieren

Das Beispiel in Abschnitt 7.5.2 zeigt, dass der Einsatz von `htmlspecialchars()` eine sehr effektive Methode ist, den Inhalt zu schützen. Wie Sie in Kapitel 6 gesehen haben,

lässt `htmlspecialchars()` standardmäßig einfache Anführungszeichen unverändert. Sicherheitshalber sollten Sie deshalb als zweiten Parameter `ENT_QUOTES` übergeben, damit alle Anführungszeichen behandelt werden.

```
echo htmlspecialchars($eingabe, ENT_QUOTES);
```

Tipp

Das ist besonders wichtig, wenn Sie eine Eingabe wieder in ein Formularfeld schreiben und selbst einfache Anführungszeichen zur Begrenzung von Attributwerten in (X)HTML benutzen:

```
<input type="text" name="nachname" value='<?php echo
htmlspecialchars($eingabe, ENT_QUOTES); ?>'>
```

`strip_tags()` lässt sich ebenfalls zur Bereinigung von Eingaben aus Formularen einsetzen. Damit werden möglicherweise eingegebene (X)HTML-Tags ganz gelöscht und nicht angezeigt. Das ist jedoch nicht immer erwünscht – bei einem Forum, in dem es um Fragen zu (X)HTML geht, würden dadurch die meisten Beiträge sinnlos.

Wenn Sie `strip_tags()` benutzen, sollten Sie zusätzlich `htmlspecialchars()` einsetzen, denn `strip_tags()` entfernt nur mögliche Tags, lässt aber alle Anführungszeichen unverändert. Und je nachdem, in welchem Kontext die Ausgabe der Inhalte erfolgt, kann es durch eingegebene Anführungszeichen zu Problemen kommen.

```
htmlspecialchars(strip_tags($eingabe), ENT_QUOTES);
```

Exkurs: Bestimmte (X)HTML-Tags erlauben

Problematisch wird es, wenn man den Benutzern bestimmte (X)HTML-Befehle zur Strukturierung oder aber beispielsweise das Einfügen von Links erlauben möchte. Zwar bietet `strip_tags()` als zweiten Parameter die Angabe von erlaubten Tags an, aber das ist nicht unproblematisch. Dies aus zwei Gründen:

- Sie können bei `strip_tags()` nur festlegen, welche (X)HTML-Elemente erlaubt sind, aber nicht die erlaubten Attribute beschränken. JavaScript-Code kann beispielsweise leicht über Eventhandler wie `onclick()` in beliebigen Elementen ergänzt werden.
- JavaScript-Code ist an mehr Stellen möglich, als man so gemeinhin annimmt: In manchen Browsern in den `src`-Attributen von `img`-Elementen und beim Internet Explorer beispielsweise auch bei CSS-Angaben.

Wenn Sie bestimmte (X)HTML-Tags erlauben möchten, brauchen Sie eine aufwändigere Lösung. Eine eigens für diesen Einsatzzweck entwickelte Bibliothek ist beispielsweise HTML Purifier (*http://htmlpurifier.org/*).

7.6.2 Input prüfen

Das Maskieren des Outputs ist relativ gradlinig. Bei der Überprüfung des Inputs aus den Formulardaten wird es ein bisschen komplizierter, weil die Anwendungen so unterschiedlich sind. Je nachdem, nach welchen Daten Sie fragen und was Sie mit den eingegebenen Daten machen wollen, gibt es ganz verschiedene Ansätze und Sie müssen mehr oder weniger Aufwand bei der Prüfung der Inhalte betreiben.

Bei allen Formulardaten sollten Sie vor dem Zugriff per `isset()` prüfen, ob die entsprechende Variable überhaupt existiert. Dies gilt auch für Radiobuttons oder Checkboxen, auch wenn eine von ihnen beim Laden des Formulars standardmäßig ausgewählt ist.

Außerdem bietet es sich bei Textfeldern an, mit `empty()` zu überprüfen, ob überhaupt etwas drin steht. Wenn Sie `empty()` einsetzen, können Sie sich die Überprüfung mit `isset()` sparen, da `empty()` bei nicht gesetzten Variablen ebenfalls `false` zurückgibt – ohne dass Sie eine Notice erhalten. Falls das Etwas nur ein Leerzeichen ist, ist das natürlich mager. Hier hilft `trim()`: Es entfernt Leerzeichen am Anfang und am Ende eines Strings.

Funktionen wie `trim()` – aber auch `htmlspecialchars()` oder `strip_tags()` –funktionieren nicht mit Arrays. Zur Überprüfung, ob etwas ein Array ist, verwenden Sie `is_array()`. Entsprechend können Sie mit `is_string()` prüfen, ob etwas ein String ist.

Zur Überprüfung der Länge des Inputs bei Textfeldern können Sie die Funktion `strlen()` verwenden. Über `substr()` lassen sich darüber hinausgehende Zeichen entfernen.

Wenn für ein bestimmtes Formularfeld nur bestimmte Werte erwünscht sind – das heißt bei Auswahllisten, Checkboxen und Radiobuttons, sollten Sie sicherstellen, dass wirklich auch nur diese und nicht etwas anderes eingegeben wurde. Hierfür erstellen Sie am besten ein Array mit den möglichen Werten und prüfen per `in_array()`, ob der angegebene Wert im Array vorhanden ist. Das folgende Beispiel führt das für eine Auswahlliste vor:

Listing 7.16: Jetzt wird geprüft, ob das gewählte Thema in der Themenliste vorhanden ist (formular_auswahlliste_pruef.php).

```
01 <?php
02 $themen = array("HTML", "CSS", "JavaScript", "PHP");
03 ?>
04 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="get">
05 Ihr Nachname: <br />
06 <input type="text" name="nachname" size="20" maxlength="30" />
07 <br />
08 Themen: <br />
09 <select name="thema">
10 <?php
```

```
11 foreach ($themen as $el) {
12   echo "<option value='$el'>$el</option>\n";
13 }
14 ?>
15 </select>
16 <br />
17 <input type="submit" value="Abschicken" />
18 </form>
19 <?php
20 if (!empty($_GET["nachname"]) && is_string($_GET["nachname"])) {
21   echo "Ihre Eingaben<br />\n";
22   echo "Name: " . htmlspecialchars($_GET["nachname"]) . "<br />\n";
23   if (isset($_GET["thema"]) && (in_array($_GET['thema'], $themen))){
24     echo "Das gewählte Thema: {$_GET['thema']}<br />\n ";
25    }
26 }
27 ?>
```

Die entscheidenden Stellen sind fett hervorgehoben: Zuerst wird ein Array mit den möglichen Werten erstellt (Zeile 2). Innerhalb des Formulars werden in einer `foreach`-Schleife die einzelnen `option`-Elemente basierend auf dem Array erstellt (Zeile 11–13). Bei der Ausgabe des Themas wird geprüft, ob das gewünschte Thema in der Themenliste vorhanden ist (Zeile 23). Nur dann wird es ausgegeben (Zeile 24).

Beim anderen Formularfeld wurde zumindest mit `is_string()` (Zeile 20) sichergestellt, dass es sich um einen String handelt und bei der Ausgabe wurde dieser String mit `htmlspecialchars()` behandelt.

Tipp

Dass die `option`-Elemente hier ebenfalls basierend auf dem Array `$themen` automatisch erstellt werden, macht Anpassungen einfacher: Sie müssen nur das Array ändern, wenn weitere Themen aufgenommen werden sollen.

In manchen Feldern sind nur Zahlen erwünscht – wenn Sie beispielsweise das Alter eines Benutzers erfragen wollen. Verwenden Sie, um zu prüfen, ob der Inhalt eines Formularfelds eine Zahl ist, die Funktion `is_numeric()`. Eine Alternative ist, den Wert mit `(int)` in einen Integer zu verwandeln.

Prinzipiell gibt es bei Prüfungen zwei Herangehensweisen: die Whitelist- und die Blacklist-Prüfung. Bei der Whitelist-Prüfung prüfen Sie, ob nur die erlaubten Zeichen/Inhalte vorkommen. Bei der Blacklist-Prüfung prüfen Sie hingegen auf die Existenz von nicht-erlaubten Zeichen. Wo immer es möglich ist, ist die Whitelist-Prüfung vorzuziehen, da die Gefahr besteht, dass man bei einer Blacklistprüfung nicht alle gefährlichen Zeichen berücksichtigt.

Achtung

`is_int()` oder `is_float()` eignen sich in diesem Fall nicht, da der Inhalt eines Formularfelds intern von PHP immer als String behandelt wird.

Schwieriger wird es, wenn Sie weitere Einschränkungen als die oben geschilderten machen möchten. Hier brauchen Sie reguläre Ausdrücke oder können auch auf die filter-Erweiterung zurückgreifen. Nicht banal ist z. B. die Überprüfung, ob eine E-Mail-Adresse korrekt ist. Ob eine E-Mail-Adresse korrekt ist, lässt sich eigentlich nur dadurch feststellen, dass man an die entsprechende Adresse eine E-Mail sendet und eine Antwort erhält. Diese Art der Überprüfung kann oder soll in vielen Fällen nicht so durchgeführt werden. Dann kann nur eine formale Prüfung verhindern, dass jemand aus Versehen seine Adresse eingibt oder seinen Namen. Eine nicht vorhandene, aber syntaktisch korrekte E-Mail-Adresse wie `dagobert@entenhausen.de` filtern Sie damit nicht aus.

Tipp

Wie E-Mail-Adresse aussehen können, definiert RFC 2822 (*http://tools.ietf.org/html/rfc2822*). Eine Implementation einer Prüfung von E-Mail-Adressen auf dieser Basis bietet ein PHP-Parser unter *http://code.iamcal.com/php/rfc822/*.

7.6.3 Variablen prüfen mit der Erweiterung filter

Die Erweiterung filter ist seit PHP Version 5.2 direkt integriert und hilft beim Filtern von Input. In der Ausgabe von `phpinfo()` sehen Sie, ob die Erweiterung `filter` aktiviert ist.

`filter` beinhaltet einige mehrere Funktionen, die alle mit `filter` beginnen und über Konstanten gesteuert werden. `filter_var()` benutzen Sie, um einen Filter anzuwenden. Als ersten Parameter übergeben Sie die Variable, die Sie filtern möchten, als zweiten Parameter, wie sie gefiltert werden soll.

Ob etwas eine E-Mail-Adresse sein kann, prüfen Sie beispielsweise so:

```
filter_var($email, FILTER_VALIDATE_EMAIL)
```

Im Erfolgsfall gibt `filter_var()` `false` zurück, ansonsten den gefilterten String. Damit lässt sich dies gut in eine `if`-Überprüfung einbauen:

Listing 7.17: Überprüfung einer E-Mail über die filter-Erweiterung (filter.php)

```
$email = "ich@mir.de";
if (filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
  echo "E-Mail $email nicht gültig";
} else {
  echo "E-Mail gültig";
}
```

Zum Filtern von ganzen Zahlen gibt es `FILTER_VALIDATE_INT`:

```
filter_var($zahl, FILTER_VALIDATE_INT);
```

Über einen dritten Parameter können Sie weitere Optionen übergeben und beispielsweise festlegen, in welchem Bereich sich die Zahl befinden soll. Im folgenden Beispiel wird überprüft, ob sich eine Zahl zwischen 3 und 99 befindet.

Listing 7.18: Sicherstellen, dass sich Zahlen in einem bestimmten Bereich bewegen (filter_zahl.php)

```
$min = 3;
$max = 99;
$test = 4;
$optionen = array(
  "options" => array(
                    "min_range"=>$min,
                    "max_range"=>$max)
);
var_dump(filter_var($test, FILTER_VALIDATE_INT, $optionen));
```

Sie sehen, wie hier die weiteren Optionen als dritter Parameter an die Funktion `filter_var()` übergeben werden. Die Optionen werden in Form eines verschachtelten Arrays angegeben mit einem Unterelement mit Namen `options`, das selbst wiederum zwei Elemente mit den Schlüsseln `min_range` und `max_range` enthält.

Da sich die Beispielzahl 4 innerhalb des angegebenen Bereichs – 3 und 99 – befindet, wird sie zurückgegeben.

Hinweis

Das war eine kleine Auswahl aus den wichtigsten Optionen zur Benutzung der `filter`-Erweiterung. Neben den hier vorgestellten Filtern zur Überprüfung gibt es noch solche zur Bereinigung von Inhalten. Die dafür benötigten Konstanten haben in ihrem Namen anstelle von `VALIDATE` die Komponente `SANITIZE`. Mit der Konstante `FILTER_SANITIZE_EMAIL` werden beispielsweise in E-Mail-Adressen nicht erlaubte Zeichen aus dem String entfernt. Eine vollständige Auflistung aller möglichen Konstanten der filter-Erweiterung finden Sie im PHP-Manual unter *http://www.php.net/manual/de/filter.constants.php*.

7.7 Formularvalidierung mit vorausgefüllten Formularfeldern

An einem Beispiel soll jetzt gezeigt werden, wie eine Überprüfung von Formulardaten stattfinden kann. Fehlen obligatorische Inhalte, wird eine Fehlermeldung ausgegeben und das Formular erneut angezeigt. Dabei werden bei diesem erneut angezeigten Formular die vorher eingegebenen Werte wieder eingetragen sind – etwas, was Sie eigens programmieren müssen.

Das Formular enthält drei Formularfelder: eines für den Vornamen, eines für den Nachnamen und eine Auswahlliste für einen Themenwunsch. Der Vorname muss nicht angegeben werden, Nachname und Themenwunsch hingegen schon und sind deswegen mit einem Sternchen gekennzeichnet. Damit man überprüfen kann, ob ein Thema gewählt wurde, wird als erster Punkt der Auswahlliste kein Thema, sondern ... angezeigt (siehe Abbildung 7.17). Wenn ... am Schluss ausgewählt bleibt, bedeutet das, dass kein Thema gewählt wurde. Sind die obligatorischen Felder nicht ausgefüllt, erscheint eine Fehlermeldung in rot und das Formular wird erneut ausgegeben. Die vorher eingetragenen Daten stehen wieder in den Formularfeldern.

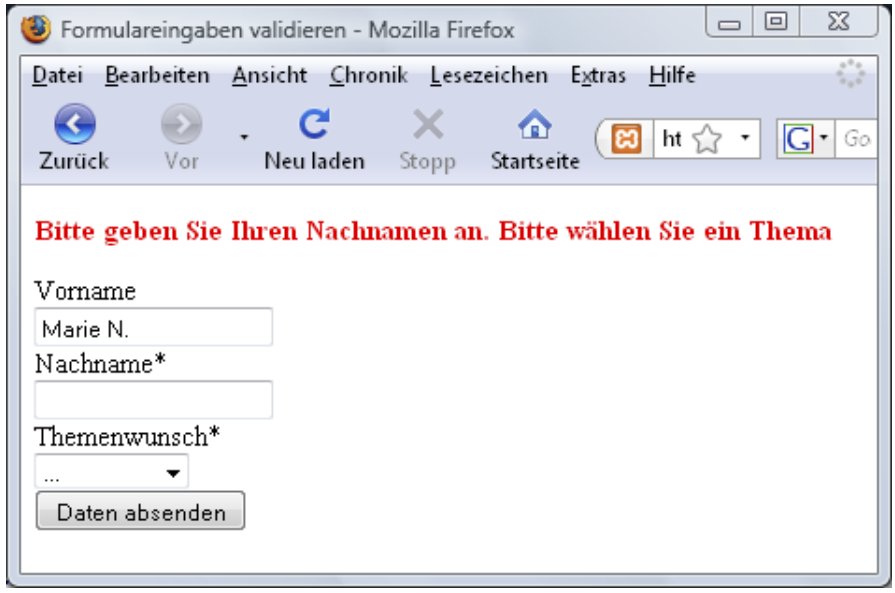


Abbildung 7.17: Meldung bei unvollständig ausgefülltem Formular

Erst einmal sehen Sie hier das gesamte Skript, danach folgen Erläuterungen.

Listing 7.19: Formular mit Validierung und Vorausfüllung (formular_validierung.php)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Formulareingaben validieren</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
  <style type="text/css">
    .fehler { color: red; font-weight: bold; }
   </style>
</head>
<body>
<?php
$themen = array("...", "HTML", "CSS", "JavaScript", "PHP");
if (isset($_POST["gesendet"])) {
  formverarbeiten();
} else {
  formausgeben();
}
function formausgeben($vorname= "", $nachname ="", $thema = "", $fehler="")
{
  global $themen;
  if (!empty($fehler)) {
    echo "<p class='fehler'>$fehler</p>";
  }
?>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);
    ?>">
Vorname <br />
<input type="text" name="vorname" size="20" maxlength="30" value="<?php echo
    htmlspecialchars($vorname); ?>" />
<br />
Nachname* <br />
<input type="text" name="nachname" size="20" maxlength="30" value="<?php echo
    htmlspecialchars($nachname); ?>" />
<br />
Themenwunsch* <br />
<select name="thema">
<?php
  foreach ($themen as $el) {
    if ($el == $thema) {
      $gew = " selected='selected' ";
    } else {
      $gew = "";
    }
    echo "<option value='$el' $gew>$el</option>\n";
   }
?>
</select>
```

```
46 <br />
47 <input type="submit" name="gesendet" />
48 <?php
49 }
50 function formverarbeiten()
51 {
52   global $themen;
53   isset($_POST["nachname"]) && is_string($_POST["nachname"]) ? $nachname =
       trim($_POST["nachname"]) : $nachname= "";
54   isset($_POST["vorname"])  && is_string($_POST["vorname"])  ? $vorname =
       trim($_POST["vorname"]) : $vorname= "";
55   isset($_POST["thema"]) && is_string($_POST["thema"])? $thema = $_POST["thema"] :
       $thema = "";
56   $fehler = "";
57   if (empty($nachname)) {
58     $fehler = "Bitte geben Sie Ihren Nachnamen an. ";
59   }
60   if (!in_array($thema, $themen) || $thema == "...") {
61     $fehler .= "Bitte wählen Sie ein Thema";
62   }
63   if (strlen($fehler) > 0) {
64     formausgeben($vorname, $nachname, $thema, $fehler);
65   } else {
66     echo "Vielen Dank für Ihre Eingaben";
67   //mail versenden;
68   }
69 }
```

Die Auswahl der Themen läuft über eine Auswahlliste. Die möglichen Inhalte legt das Array `$themen` in Zeile 13 fest.

Die Zeilen 14 bis 18 beinhalten kurz und knapp die Hauptlogik des gesamten Skripts:

```
14 if (isset($_POST["gesendet"])) {
15   formverarbeiten();
16 } else {
17   formausgeben();
18 }
```

Wenn `$_POST["gesendet"]` gesetzt ist, also das Formular abgesendet wurde, soll die Funktion `formverarbeiten()` aufgerufen werden, ansonsten die Funktion `formausgeben()`. Wie Sie später sehen werden, wird `formausgeben()` ebenfalls im Falle eines Fehlers aufgerufen.

In Zeile 19 wird die Funktion `formausgeben()` erstellt, die das Formular ausgibt. Sie erwartet vier Parameter, die die Inhalte der Formularfelder bei einer erneuten Anzeige nach einer Fehlermeldung sowie die Fehlermeldung selbst beinhalten, aber defaultmäßig auf einen Leerstring gesetzt werden. Dies ist wichtig, weil `formausgeben()` auch bei einem Fehler aufgerufen wird, damit in diesem Fall die bereits eingetragenen Werte nicht verloren gehen, sondern erneut eingetragen werden können.

Zeile 21 sorgt über `global` dafür, dass auf das Array `$themen`, das außerhalb der Funktion definiert ist, zugegriffen werden kann.

Falls eine Fehlermeldung übergeben wurde, wird diese in Zeile 23 ausgegeben.

Ab Zeile 26 wird das Formular erstellt. Wir sehen uns die Funktion `formausgeben()` gleich noch einmal an, um genauer zu betrachten, was passiert, wenn diese nach einem Fehler aufgerufen wird.

In Zeile 50 wird die Funktion `formverarbeiten()` definiert. In der Funktion wird zuerst überprüft, ob alle Variablen gesetzt sind und ob es sich bei ihnen um Strings handelt – denn es werden hier Strings erwartet. Wenn ja, werden mögliche Leerzeichen entfernt, sonst werden die Variablen auf Leerstrings gesetzt.

Sehen wir uns das einmal an einem Beispiel an:

```
53   isset($_POST["nachname"]) && is_string($_POST["nachname"]) ? $nachname =
     trim($_POST["nachname"]) : $nachname= "";
```

Zur Überprüfung kommt der ternäre Operator zum Einsatz: Sind die beiden ersten Bedingungen vor dem Fragezeichen wahr, wird in `$nachname` der über `trim()` von Leerzeichen bereinigte String `$_POST["nachname"]` gespeichert, ansonsten `$nachname` auf den Leerstring gesetzt.

In Zeile 57 findet eine Überprüfung statt, ob die Variable `$nachname` leer ist und in diesem Fall eine passende Fehlermeldung in der Variable `$fehler` gespeichert.

In Zeile 60 findet die Überprüfung des Themas statt. Es muss ein Element des Arrays `$themen` sein und darf außerdem nicht dem `...` (die erste Option, die standardmäßig angezeigt wird) entsprechen. Ansonsten wird die Variable `$fehler` um diese weitere Fehlermeldung ergänzt.

Falls der String in der Variable `$fehler` länger ist als 0, ist ein Fehler aufgetreten (Zeile 63). Dann wird die Funktion `formausgeben()` erneut aufgerufen, ihr aber die Variablen `$vorname`, `$nachname`, `$thema` und `$fehler` übergeben, damit diese in das Formular eingetragen werden können.

Ansonsten (Zeile 65) wird die Meldung »Vielen Dank für Ihre Eingaben« ausgegeben. An dieser Stelle würde auch die weitere Verarbeitung stattfinden – beispielsweise die Daten per Mail zu versenden (Abschnitt 7.8) oder in einer Datenbank zu speichern (Kapitel 11).

Sehen wir uns noch einmal an, wie die Funktion `formausgeben()` funktioniert, wenn sie nach einem Fehler aufgerufen wird: Dann sind die Variablen `$vorname`, `$nachname`, `$thema` und `$fehler` ja gefüllt. Da jetzt ein Fehler aufgetreten ist, beinhaltet die Variable `$fehler` die Fehlermeldung und wird in Zeile 23 ausgegeben. Die Angabe `class='fehler'` verweist auf die CSS-Formatierung der Klasse in Zeile 9 und sorgt dafür, dass der Text rot und fett ist.

Bei der Erstellung der Textfelder in Zeile 28 und 31 werden die vorher eingetragenen Werte bei `value` ausgegeben, so beispielsweise beim Feld `vorname`:

```
28 <input type="text" name="vorname" size="20" maxlength="30" value="<?php echo
     htmlspecialchars($vorname); ?>" />
```

Bei der Auswahlliste geht das etwas anders: Damit die vorher ausgewählte Option wieder ausgewählt ist, muss diese mit `selected='selected'` gekennzeichnet werden. Bei der Erstellung der Auswahlliste über eine `foreach`-Schleife wird bei der Ausgabe jedes Punktes überprüft, ob es sich um das aktuelle Thema handelt und in diesem Fall in der Variable `$gew = "selected='selected' "` gespeichert, ansonsten ein Leerstring. Dann erst wird die Option ausgegeben:

```
34 <select name="thema">
35 <?php
36   foreach ($themen as $el) {
37     if ($el == $thema) {
38       $gew = " selected='selected' ";
39     } else {
40       $gew = "";
41     }
42     echo "<option value='$el' $gew>$el</option>\n";
43   }
44 ?>
45 </select>
```

Achtung

Bei diesem Beispiel wird davon ausgegangen, dass Magic Quotes deaktiviert sind. Ist das nicht der Fall, müssen Sie zusätzlich `stripslashes()` anwenden.

Tipp

Zusätzlich zur Überprüfung per PHP können Sie eine Überprüfung per JavaScript durchführen. Der JavaScript-Code wird im Browser ausgeführt. Damit wird die Überprüfung durchgeführt, bevor die Formulardaten abgesendet werden, und der Benutzer erhält sofort ein Feedback, was aus Gründen der Usability positiv ist. Der Nachteil an der Prüfung per JavaScript ist jedoch, dass Sie sich nicht auf sie verlassen können: JavaScript kann im Browser jederzeit deaktiviert werden, deswegen ist es sinnvoll die Überprüfung `mit JavaScript und mit PHP` durchzuführen. Ein Beispiel, wie sich eine Überprüfung per JavaScript im Beispiel integrieren lässt, finden Sie im Listing *formular_validierung_plus_js.php* bei den Listings auf der Webseite zum Buch unter http://www.addison-wesley.de/9783827327239.html.

7.8 Formulardaten per E-Mail versenden

Die in ein Formular eingetragenen Werte sollen häufig per Mail versendet werden.

7.8.1 E-Mail versenden – Grundlagen

Eine E-Mail per PHP zu versenden, ist sehr einfach. Hierzu gibt es die Funktion `mail()`. Sie erwartet drei Parameter: den Adressaten, den Betreff und den eigentlichen Text.

```
mail("ich@mir.de", "Hallo von PHP", "Eine erste E-Mail");
```

Hinweis

Eine E-Mail setzt sich prinzipiell aus zwei Bestandteilen zusammen: aus dem Header mit den Kopfzeilen und dem Body mit dem eigentlichen Inhalt der Nachricht. Adressat und Betreff gehören zum Header, der Textinhalt ist Teil des Bodys.

Im Header sind neben Adressat und Betreff viele weitere Informationen möglich, wie der Absender (`From`), Kopieempfänger (`Cc` und `Bcc`), an wen die Antwort gehen soll (`Reply-To`) oder auch die Priorität (`X-Priority`) und viele mehr. Mailprogramme zeigen üblicherweise nicht alle Header an, das kann man aber bei Bedarf bei einzelnen Mails auch ändern – bei Thunderbird beispielsweise über ANSICHT/KOPFZEILEN/ALLE, wenn man gerade eine Mail einzeln aufgerufen hat.

Bei der `mail()`-Funktion ist ein vierter Parameter mit zusätzlichen Headern möglich: Beispielsweise mit der Absenderadresse über `From: ich@example.com`:

```
mail("ich@mir.de", "Hallo von PHP", "Eine erste E-Mail", "From: ich@example.com");
```

Weitere Header können angegeben werden – etwa an wen eine Kopie einer Mail gehen soll (`Cc`) oder an wen eine unsichtbare Kopie gehen soll `Bcc`. Diese Header müssen Sie durch `\r\n` trennen.

```
mail("newuser@localhost", "Hallo von PHP", "Eine erste E-Mail", "FROM:
ich@example.com\r\nCc: du@example.com");
```

Mehrere Adressen bei `Cc` oder `Bcc` geben Sie durch Komma getrennt an.

`mail()` liefert `true` zurück, wenn die Mail zur Auslieferung akzeptiert wurde, das bedeutet, Sie können `mail()` in eine Überprüfung mit `if()` einbauen. Allerdings heißt der Rückgabewert `true` nicht, dass die Mail auch zugestellt werden konnte.

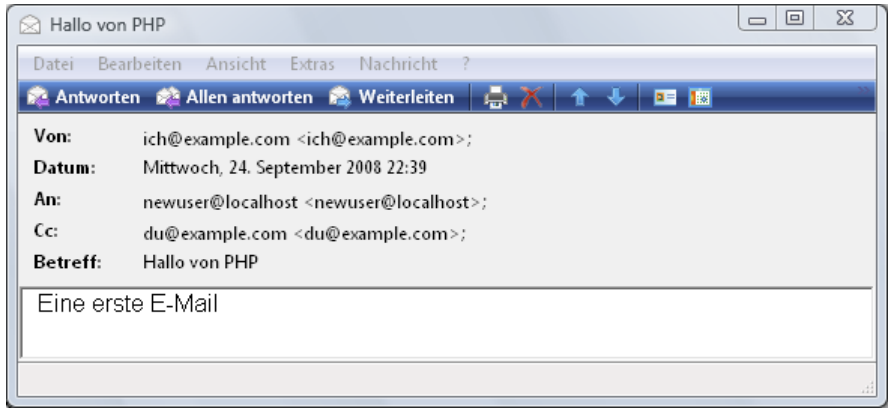


Abbildung 7.18: Die Beispiel-E-Mail mit weiteren Headern im Mailprogramm Windows Mail

Exkurs: Mails lokal verschicken

Das Verwenden der `mail()`-Funktion ist wirklich einfach und unproblematisch. Das lokale Testen ist jedoch eher mühsam, denn Sie benötigen einen richtig konfigurierten Mailserver. Unter Linux brauchen Sie einen SMTP-Server, bei Windows können Sie testweise auch auf den bereits bei XAMPP installierten Mailserver Mercury zugreifen. Erste Hinweise zu Mercury finden Sie in der Benutzeroberfläche von XAMPP unter dem Punkt MERCURY.

7.8.2 Daten aus Formularen per E-Mail versenden

Nun zur Versendung von Formulardaten per Mail. Sie können innerhalb von Betreff und Nachricht jetzt natürlich auch Variablen mit den Inhalten aus dem Formular integrieren.

Angenommen, `$anrede`, `$name`, `$email`, `$kommentar` stammen aus einem Kontaktformular, dann lässt sich eine Nachricht folgendermaßen zusammensetzen.

```
$nachricht = "$anrede $name ($email) hat folgendes Anliegen: $kommentar";
mail("ich@mir.de", "Nachricht vom Kontaktformular", $nachricht, "From:
ich@example.com");
```

Was Sie allerdings *vermeiden* sollten, ist eine aus einem Formular übernommene Absenderadresse direkt als zusätzlichen Header hinter `From:` zu ergänzen:

```
/* das sollten Sie nicht ohne genaue Prüfung der Mail-Adresse machen */
$nachricht = "$anrede $name hat folgendes Anliegen: $kommentar";
mail("ich@mir.de", "Nachricht vom Kontaktformular", $nachricht, "From: $email" );
```

Der Grund: Jemand könnte in das entsprechende Formularfeld für die E-Mail-Adresse zusätzliche E-Mail-Header integrieren und das Formular als Spamschleuder missbrauchen. Dies ist möglich durch eine folgendermaßen konstruierte Eingabe in das Formularfeld für die E-Mail-Adresse:

```
ich@mir.de\r\nBcc: zusgespammt@example.com, dirauch@example.com,
auchdumeinsohn@example.com
```

Die einfache Methode, um das zu verhindern, ist es, als `From`-Header eine feste Adresse anzugeben und sie nicht aus einem Formularfeld zu übernehmen. Die andere Möglichkeit besteht darin, die E-Mail-Adresse zu validieren – siehe hierzu Abschnitt 7.6.3.

Mit `mail()` versenden Sie auf diese Art normale Textmails – also sollten Sie darauf achten, keine (X)HTML-Tags einzuschließen und Zeilenumbrüche in der Nachricht über `\n` zu realisieren.

7.8.3 HTML-Mails verschicken

Wenn Sie HTML-Mails verschicken möchten oder müssen, sollten Sie dies über das PEAR-Paket `Mail_Mime` realisieren (*http://pear.php.net/package/Mail_Mime*) – das ist wesentlich komfortabler. Dieses erlaubt es, Mails gleichzeitig als Text- und HTML-Mails zu versenden und Attachments mitzuschicken. Dabei arbeiten zwei PEAR-Pakete zusammen: Das Paket `Mail_Mime` übernimmt zuverlässig die Erstellung der MIME-E-Mail, für das Versenden wird hingegen das PEAR-Paket `Mail` benutzt.

Um die beiden Pakete zu installieren, geben Sie bei installiertem Pear-Paket-Manager in die Eingabeaufforderung Folgendes ein:

```
pear install Mail
```

und dann noch

```
pear install Mail_Mime
```

Dann werden die benötigten Dateien heruntergeladen und gespeichert.

Tipp

Wenn der PEAR-Paket-Manager bei Ihnen nicht vorhanden ist, können Sie sich auch einfach die benötigten Dateien unter *http://pear.php.net/package/Mail_Mime/download* herunterladen, damit entfallen aber die komfortablen Update-Möglichkeiten über PEAR.

Sehen wir uns ein Beispiel für die Versendung einer einfachen Text/HTML-Mail mit Attachment an.

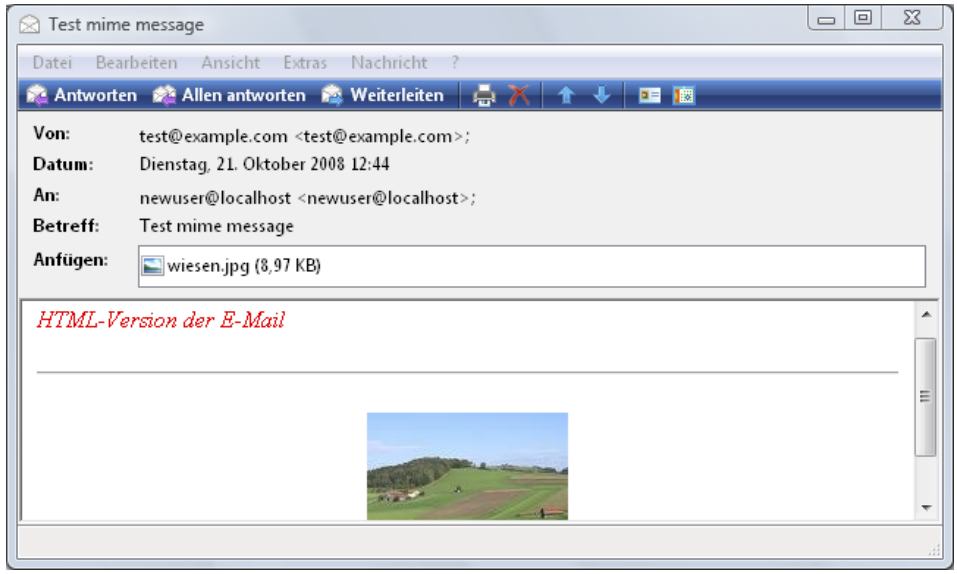


Abbildung 7.19: HTML-Version der E-Mail in Windows Mail

Sie werden sehen, dass `Mail_Mime` und `Mail` beide objektorientiert programmiert sind.

Listing 7.20: Eine MIME-Mail mit Mail_mime erstellen (mail_mime.php)

```
01 error_reporting(0);
02 require_once "Mail.php";
03 require_once "Mail/mime.php";
04 $text = "Textversion der E-Mail";
05 $html = "<html><body><p style='color:red;font-style:italic'>HTML-Version der E-
     Mail</p></body></html>";
06 $datei = "wiesen.jpg";
07 $mailheaders = array(
08               "From"    => "test@example.com",
09               "Subject" => "Test mime message"
10               );
11 $mime = new Mail_mime();
12 $mime->setTXTBody($text);
13 $mime->setHTMLBody($html);
14 $mime->addAttachment($datei, "image/jpeg");
15 $body = $mime->get();
16 $headers = $mime->headers($mailheaders);
17 $mail =& Mail::factory("mail");
18 $mail->send("newuser@localhost", $headers, $body);
```

In Zeile 1 wird erst einmal die Anzeige der Fehlermeldungen ausgeschaltet, weil Sie ansonsten Strict- und Deprecated-Meldungen erhalten.

Zeile 2 und Zeile 3 binden die benötigten Dateien ein.

Jetzt werden mehrere Variablen definiert, die die Inhalte für die E-Mail bereitstellen. Die Variable `$text` nimmt den normalen Textinhalt der Mail auf, entsprechend die Variable `$html` die HTML-Version. In der Variable `$datei` wird der Pfad zu einer lokal vorhandenen Datei angegeben, die als Attachment mitgesendet werden soll. Im Beispiel ist es eine Bilddatei, die sich im selben Ordner wie das Skript befindet.

In Zeile 7 wird ein Array mit Headern definiert: im Beispiel nur der Absender (`From`) und der Betreff (`Subject`).

Zeile 11 erstellt eine neue Instanz der Klasse `Mail_mime`.

Jetzt erfolgt der Aufruf mehrerer definierter Methoden der Klasse `Mail_mime`: `setTXTBody()` definiert den Text, der als Textmail versendet werden soll, entsprechend `setHTMLBody()` den Inhalt für die HTML-Version. Die Methode `addAttachment()` erwartet als ersten Parameter den Namen der Datei, die als Attachment versendet werden soll, und als zweiten den MIME-Typ des Attachments.

Die Methode `get()` erstellt die Mail und gibt diese zurück. Sie sollten `get()` aufrufen, nachdem Sie die Bestandteile der E-Mail (Text, HTML, Attachment) hinzugefügt haben. Die Methode `headers()` baut entsprechend die E-Mail-Header zusammen und erwartet das vorher angelegte Array mit den Headern.

Wie bereits erwähnt, wird nicht die Klasse `Mail_mime`, sondern die PEAR-Klasse `Mail` zum Versenden der Mail eingesetzt. Zwei Methoden werden von der Klasse `Mail` verwendet, die `factory()`- und die `send()`-Methode. Zeile 17 erstellt eine neue Instanz der Klasse Mail über die `factory()`-Methode.

Schließlich übergeben Sie der `send()`-Methode den Adressaten, die erstellten Header und den erstellten Body zur Versendung der Mail.

7.9 Dateien hochladen

Über PHP können Sie auch vom Benutzer hochgeladene Dateien entgegennehmen und verarbeiten.

7.9.1 Dateiupload: Grundlegendes

Dafür müssen Sie erst einmal das Formular zum Dateiupload erstellen:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post"
  enctype="multipart/form-data">
Datei: <br />
<input type="hidden" name="MAX_FILE_SIZE" value="30000" />
<input type="file" name="datei" /><br />
<input type="submit" value="Hochladen" />
</form>
```

Wichtig ist, dass Sie beim Dateiupload als Übertragungstyp POST und nicht GET benutzen. Zusätzlich notwendig ist `enctype="multipart/form-data"` im `form`-Starttag.

Das Feld zum Hochladen einer Datei ist ein `input`-Element vom Typ `file`, dem Sie außerdem wie gewohnt einen Namen geben. Der Browser erzeugt Ihnen dann automatisch die benötigte Durchsuchen-Schaltfläche. Wenn der Benutzer daraufklickt, öffnet sich das Menü zum Dateihochladen.

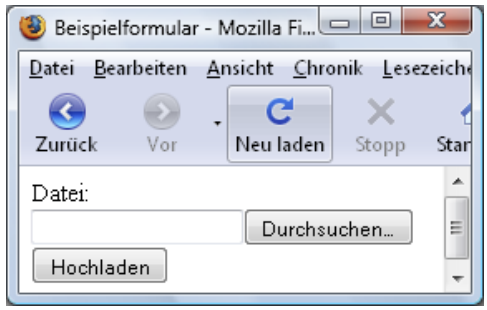


Abbildung 7.20: Ein Formular zum Hochladen einer Datei

Mit dem versteckten Formularfeld mit Namen `MAX_FILE_SIZE` können Sie eine maximal akzeptierte Dateigröße vorgeben. Der bei `value` stehende Wert wird in Byte angegeben. Dieses versteckte Feld muss vor dem Datei hochladen-Feld stehen, damit es von PHP korrekt ausgewertet wird. Eine praktische Angabe – nur leider kann man sich darauf nicht verlassen. Sie haben in Abschnitt 7.5.3 gesehen, wie man Formulare manipulieren kann – und das gilt gleichermaßen für dieses versteckte Feld.

So weit schön und gut – Sie können damit schon eine Datei hochladen, jedoch wird diese standardmäßig sofort gelöscht. Fehlt also noch der PHP-Code, der dafür sorgt, dass die Datei entgegengenommen und an geeignete Stelle kopiert wird. Dafür benötigen Sie bestimmte Informationen über die hochgeladene Datei, die Ihnen im assoziativen Array `$_FILES["name-des-Inputfelds"]` zur Verfügung stehen. Diese können Sie sich einmal ausgeben lassen (Listing 7.21).

Listing 7.21: Informationen über eine hochgeladene Datei anzeigen lassen (datei_upload_info.php)

```
if (isset($_FILES["datei"])) {
  foreach ($_FILES["datei"] as $k => $v) {
    echo "$k: $v <br />\n";
  }
}
```

- `$_FILES["datei"]["name"]` enthält den Namen der Datei auf dem System des Anwenders.
- `$_FILES["datei"]["type"]` beinhaltet den MIME-Type der Datei, so wie der Browser ihn an den Server gesendet hat. (Ist also auch nicht sicher.)

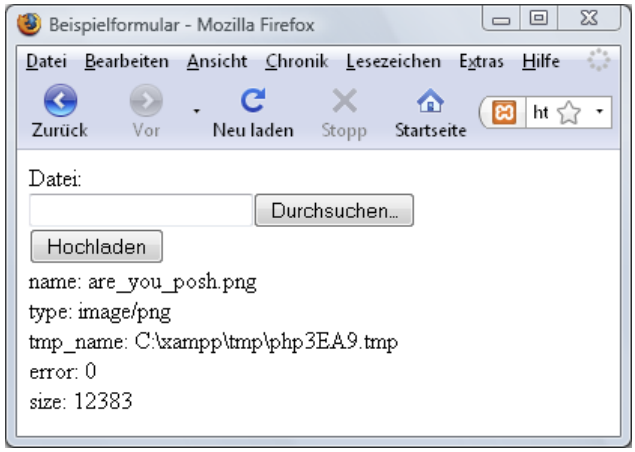


Abbildung 7.21: Im Beispiel wurde eine PNG-Datei hochgeladen.

Hinweis

MIME steht für Multipurpose Internet Mail Extensions und ist ein Standard zur Bezeichnung verschiedener Medientypen. Eine Auflistung möglicher MIME-Typen finden Sie unter *http://www.iana.org/assignments/media-types*.

- `$_FILES["datei"]["tmp_name"]` ist der temporäre Dateiname inklusive Pfad, wo die hochgeladene Datei temporär gespeichert ist. Von dort muss die Datei später abgeholt und in das gewünschte Verzeichnis kopiert werden.
- `$_FILES["datei"]["error"]` beinhaltet den Fehlercode. Wenn alles geklappt hat, ist der Fehlercode 0.
- `$_FILES["datei"]["size"]` gibt die Größe der hochgeladenen Datei in Bytes an und ist zuverlässiger als das eben erwähnte versteckte Feld.

Jetzt müssen Sie die Datei aus dem temporären Verzeichnis in das gewünschte Zielverzeichnis kopieren. Dafür ist die Funktion `move_uploaded_file()` gedacht. Sie überprüft gleichzeitig auch, ob es sich bei der Datei wirklich um eine hochgeladene Datei handelt.

`move_uploaded_file()` gibt `false` zurück, wenn das zu verschiebende Dokument keine hochgeladene Datei ist oder wenn das Verschieben nicht geklappt hat, weil beispielsweise das Verzeichnis, in das die Datei hochgeladen werden soll, nicht beschreibbar ist.

Achtung

Deswegen sollten Sie unbedingt immer diese Funktion verwenden und nicht die Funktion `copy()`.

Tipp

Wenn Sie den Dateiupload bei Ihrem Provider testen, so können Sie das Verzeichnis dort über Ihr FTP-Programm beschreibbar machen. Im Allgemeinen geht das über einen Rechtsklick auf den Ordner und durch Setzen der entsprechenden Eigenschaften. Mehr hierzu und zu den verschiedenen möglichen Rechten neben den Schreibrechten lesen Sie in Kapitel 12.

Das folgende Listing zeigt, wie eine hochgeladene Datei in den Unterordner *upload* verschoben wird.

Listing 7.22: Die hochgeladene Datei wird in das gewünschte Verzeichnis verschoben (datei_upload.php).

```
01 if (isset($_FILES["datei"]) AND ! $_FILES["datei"]["error"]) {
02   if (move_uploaded_file($_FILES["datei"]["tmp_name"], "upload/" .
basename($_FILES["datei"]["name"]))) {
03     echo "Dateiupload hat geklappt";
04   } else {
05     echo "Dateiupload hat nicht geklappt";
06   }
07 }
```

Mit `if` wird in der ersten Zeile überprüft, ob eine Datei hochgeladen wurde und kein Fehler auftrat. In diesem Fall wird versucht, diese Datei mit `move_uploaded_file()` zu verschieben. Der Funktion `move_uploaded_file()` wird als erster Parameter der ursprüngliche Dateiname und dann das Ziel übergeben. Als Ziel wird das Verzeichnis *upload* angegeben und als Dateiname der ursprüngliche Dateiname. Da manche Browser beim ursprünglichen Dateinamen zusätzlich den vollständigen Pfad angeben, wird `basename()` eingesetzt, das eine eventuell vorhandene Pfadangabe entfernt.

7.9.2 Skript für den Bildupload

Beim Dateiupload besteht die Gefahr, dass jemand beispielsweise ein bösartiges PHP-Skript hochlädt und es danach aufruft. Es wird dann ausgeführt und damit der ganze Code, der darin steht. Damit kann jemand beliebigen Code ausführen. Wie sich so etwas verhindern lässt, soll am Beispiel eines Bildupload-Skripts gezeigt werden.

Ein erster Ansatz, das zu verhindern, geht über die Prüfung des MIME-Typs der Datei. Dieser steht in `$_FILES["datei"]["type"]`, allerdings so, wie der Browser ihn an den Server gesendet hat, und kann beliebig manipuliert werden. Besser ist es deswegen, einen anderen Weg zu gehen.

Achtung

Falls Sie doch einmal mit dem über das `$_FILES`-Array übermittelten MIME-Typ arbeiten: Beachten Sie, dass der Internet Explorer bei JPEG-Bildern `image/pjeg` liefert, andere Browser hingegen `image/jpeg`. Dieses Problem haben Sie nicht, wenn Sie den MIME-Typ über `getimagesize()` ermitteln (siehe nachfolgenden Absatz).

MIME-Typ über getimagesize() ermitteln

Sicherer ist es, wenn Sie den MIME-Typ über `getimagesize()` ermitteln. Die Funktion `getimagesize()` ermittelt – anders als der Name vermuten lässt – mehr als nur die Größe des Bildes. Es liefert ein Array mit verschiedenen Informationen zum Bild. Angenommen, Sie speichern die Rückgabe von `getimagesize()` in einem Array namens `$bildinfo`:

```
$bildinfo = getimagesize($_FILES["datei"]["tmp_name"])
```

Dann können Sie auf mehrere nützliche Informationen zugreifen, unter anderen:

- `$bildinfo[0]` beinhaltet die Breite
- `$bildinfo[1]` entsprechend die Höhe
- `$bildinfo["mime"]` gibt den MIME-Typ des Bilds zurück.
- `getimagesize()` liefert außerdem `false` zurück, wenn es sich nicht um ein Bild handelt.

Hinweis

Seit PHP 5.3 können Sie auch die integrierte Erweiterung `fileinfo` verwenden, um den MIME-Typ einer Datei zu ermitteln. `fileinfo` zieht zur Bestimmung des Dateityps eine interne Liste von Informationen über Dateitypen heran. Diese befindet sich bei Linux/Unix normalerweise unter */etc/magic*. Windows-Nutzer können eine solche Datei unter *http://gnuwin32.sourceforge.net/downlinks/file-bin-zip.php* herunterladen. Benutzt wird dann die Datei *share/file/magic* – den Pfad dazu müssen Sie angeben. Bei XAMPP unter Windows befindet sich diese Datei standardmäßig unter *\xampp \php\extras\magic.mime*.

Dann lässt sich diese Erweiterung folgendermaßen nutzen:

```
$finfo = finfo_open(FILEINFO_MIME, "/pfad/zu/magic/mime");
if (!$finfo) {
  echo "fileinfo-Datenbank konnte nicht geöffnet werden";
  exit();
}
/* MIME-Typ einer Datei ermitteln und ausgeben lassen*/
$filename = "/pfad/zur/datei";
echo finfo_file($finfo, $filename);

/* Verbindung schließen */
finfo_close($finfo);
```

Weitere Schutzvorkehrungen

Nur mit der richtigen Endung: Außerdem ist es noch zusätzlich empfehlenswert dafür zu sorgen, dass nur Dateien mit erwünschten Endungen im Upload-Verzeichnis landen – da Sie auch über PHP Bilder erstellen können (siehe Kapitel 13). Bei diesen per PHP erstellten Bildern würde `getimagesize()` ebenfalls `true` zurückliefern. Diese Bilder haben allerdings dann die Endung `.php` und könnten weiteren schadhaften Code enthalten.

Tipp

Und der Webserver muss natürlich so konfiguriert sein, dass er Dateien mit der Endung `.jpg`, `.gif` oder `.png` nicht an den PHP-Parser weiterleitet. Ansonsten könnte nämlich in einem Bild als Textkommentar PHP-Code eingebunden werden.

Um sicherzustellen, dass das Bild die gewünschte Endung hat, werden wir gleich im Skript die ursprüngliche Endung entfernen und dann eine neue Endung anhängen, die aus dem MIME-Typ ermittelt wurde.

Nur erwünschte Zeichen im Bildnamen: Außerdem sollten Sie dafür Sorge tragen, dass nur erwünschte Zeichen im Bildnamen vorkommen. Das wird im folgenden Skript durch einen regulären Ausdruck sichergestellt.

Keine bestehenden Dateien überschreiben: Wenn jemand ein Bild hochlädt, und ein gleichnamiges existiert im Zielverzeichnis, so wird es überschrieben. Auch das wird im folgenden Skript abgefangen: Wenn die entsprechende Datei schon existiert, wird `kopie_` an den Anfang des Namens ergänzt. Und das so oft, bis es keine Datei mehr unter dem angegebenen Namen im Verzeichnis gibt.

Hinweis

Da die Gefahr beim Dateiupload dadurch droht, dass jemand eine gefährliche Datei direkt aufruft, ist es eine gute Schutzmethode, die Dateien *außerhalb des Webverzeichnisses* zu speichern. Allerdings haben Sie bei Shared Hosting-Angeboten diese Möglichkeit oft nicht. Wie Sie Dateien außerhalb des Webverzeichnisses speichern, zeigt ein Artikel unter *http://www.scanit.be/uploads/php-file-upload.pdf*. Er führt außerdem weitere Punkte auf, die für eine weitergehende Sicherung von Uploads zu beachten sind.

Bildupload-Skript

Jetzt zum Bildupload-Skript, das diese Punkte berücksichtigt. Sie sehen es zuerst in seinem Gesamtzusammenhang, danach werden die einzelnen Teile besprochen:

Listing 7.23: Das Skript zum Upload von Bilddateien (datei_upload_bild.php)

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post"
enctype="multipart/form-data">
Datei: <br />
<input type="hidden" name="MAX_FILE_SIZE" value="300000" />
<input type="file" name="datei" /><br />
<input type="submit" value="Hochladen" />
</form>

<?php
if (isset($_FILES["datei"]) AND ! $_FILES["datei"]["error"]  AND
    ($_FILES["datei"]["size"] < 300000 )) {
    $bildinfo = getimagesize($_FILES["datei"]["tmp_name"]);
    if ($bildinfo === false) {
      die("kein Bild");
    } else {
      $mime = $bildinfo["mime"];
      $mimetypen = array (
        "image/jpeg" => "jpg",
        "image/gif" => "gif",
        "image/png" => "png"
```

```
19          );
20       if (!isset($mimetypen[$mime])) {
21         die("nicht das richtige Format");
22       } else {
23         $endung = $mimetypen[$mime];
24       }
25       $neuername = basename($_FILES["datei"]["name"]);
26       $neuername = preg_replace("/\.(jpe?g|gif|png)$/i", "", $neuername);
27       $neuername = preg_replace("/[^a-zA-Z0-9_-]/", "", $neuername);
28       $neuername .= ".$endung";
29       $ziel = "upload/$neuername";
30       while (file_exists($ziel)) {
31         $neuername = "kopie_$neuername";
32         $ziel = "upload/$neuername";
33       }
34        if (@move_uploaded_file($_FILES["datei"]["tmp_name"], $ziel)){
35          echo "Dateiupload hat geklappt";
36       } else {
37         echo "Dateiupload hat nicht geklappt";
38      }
39    }
40 }
41 ?>
```

Zu Beginn – Zeile 1–6 – wird das Formular erstellt.

```
01 <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post"
     enctype="multipart/form-data">
02 Datei: <br />
03 <input type="hidden" name="MAX_FILE_SIZE" value="300000" />
04 <input type="file" name="datei" /><br />
05 <input type="submit" value="Hochladen" />
06 </form>
```

In Zeile 9 werden erste grobe Überprüfungen durchgeführt:

```
09 if (isset($_FILES["datei"]) AND ! $_FILES["datei"]["error"]  AND
     ($_FILES["datei"]["size"] < 300000 )) {
```

Nur wenn das Formular abgesendet wurde, es keine Fehler beim Upload gab und die Größe den vorgegebenen Wert nicht überschreitet, wird die Überarbeitung durchgeführt.

In den nächsten Zeilen wird über `getimagesize()` überprüft, ob es sich um ein Bild handelt. Wenn nicht, wird die Verarbeitung über `die()` abgebrochen.

```
10     $bildinfo = getimagesize($_FILES["datei"]["tmp_name"]);
11     if ($bildinfo === false) {
12       die("kein Bild");
```

Hinweis

`die()` bricht die Verarbeitung eines Skripts ab. Vorher wird aber noch die in Klammern stehende Meldung ausgegeben.

Ansonsten wird der MIME-Typ ermittelt:

```
13     } else {
14       $mime = $bildinfo["mime"];
```

Nun wird ein Array erstellt, das zu den einzelnen zugelassenen MIME-Typen die Endungen definiert.

```
15       $mimetypen = array (
16         "image/jpeg" => "jpg",
17         "image/gif" => "gif",
18         "image/png" => "png"
19         );
```

Wenn der MIME-Typ der hochgeladenen Datei nicht in der Liste ist, wird mit einer Fehlermeldung abgebrochen, ansonsten die Variable `$endung` auf die entsprechende Endung gesetzt.

```
20      if (!isset($mimetypen[$mime])) {
21        die("nicht das richtige Format");
22      } else {
23        $endung = $mimetypen[$mime];
24      }
```

Jetzt wird der neue Dateiname zusammengesetzt: Als Basis dient der übergebene Name:

```
25      $neuername = basename($_FILES["datei"]["name"]);
```

Die vorhandene Endung `jpeg`, `jpg`, `gif` oder `png` wird durch nichts – d.h. durch einen leeren String – ersetzt:

```
26      $neuername = preg_replace("/\.(jpe?g|gif|png)$/i", "", $neuername);
```

Außerdem sollen nur bestimmte Zeichen im Dateinamen zugelassen werden, alle anderen werden gelöscht.

```
27      $neuername = preg_replace("/[^a-zA-Z0-9_-]/", "", $neuername);
```

Schließlich wird die Endung drangehängt:

```
28      $neuername .= ".$endung";
```

Jetzt wird das Ziel angegeben: Die Datei soll im Verzeichnis *upload* unter dem neu erstellten Namen abgespeichert werden. Ob dort eine gleichnamige Datei bereits existiert, prüft `file_exists()`. Diese von PHP vorgegebene Funktion liefert `true` zurück, wenn dort bereits eine Datei mit dem angegebenen Namen existiert. In diesem Fall wird der Name mit `kopie_` am Anfang versehen (Zeile 31). Die Überprüfung findet innerhalb einer `while`-Schleife statt, d. h. es wird so lange `kopie_` an den Dateinamen gehängt, bis keine gleichnamige Datei mehr vorhanden ist.

```
29      $ziel = "upload/$neuername";
30      while (file_exists($ziel)) {
31        $neuername = "kopie_$neuername";
32        $ziel = "upload/$neuername";
33      }
```

Schließlich wird die Datei über `move_uploaded_file()` am angegebenen Ort abgespeichert. Eventuell auftretende Fehlermeldungen werden über das `@`-Zeichen vor der Funktion abgefangen.

```
34       if (@move_uploaded_file($_FILES["datei"]["tmp_name"], $ziel)){
35         echo "Dateiupload hat geklappt";
36      } else {
37        echo "Dateiupload hat nicht geklappt";
38     }
```

In Zeile 39 folgt die schließende Klammer des `else`-Zweigs, der in Zeile 22 begonnen hat, in Zeile 40 die schließende Klammer des umfassenden `if` aus Zeile 9.

Damit beenden wir die Behandlung der Formulare, aber auch im nächsten Kapitel geht es um wichtige Webtechniken – nämlich darum, wie Sie mit Cookies und Sessions Benutzer wiedererkennen und Zustände speichern.

8 Zustände behalten über Cookies und Sessions

HTTP – das Protokoll, über das der Austausch von Daten zwischen Webserver und Client stattfindet – ist statuslos. Das bedeutet Folgendes: Ein Browser fordert vom Webserver ein bestimmtes Dokument an. Dann fordert derselbe Browser vom selben Webserver ein weiteres Dokument an. Diese beiden Vorgänge sind jedoch vollständig unabhängig voneinander, d. h. der Webserver merkt sich nicht, dass er an den Client bereits ein Dokument gesendet hat. Für manche Anwendungen braucht man aber gerade das, eine Möglichkeit, einen Anwender wiederzuerkennen. Klassisches Beispiel ist der Warenkorb in einem Shop-System, in den ein Benutzer Produkte hinzufügen kann und der diese Produkte behält, auch wenn der Benutzer die Seite wechselt. Auch für personalisierte Seiten, die einen Benutzer persönlich begrüßen und die auf ihn zugeschnittene Inhalte zeigen, braucht man eine Möglichkeit, Zustände zu speichern.

Wenn Sie ein Formular einsetzen, können Sie über versteckte Felder (`type="hidden"`) Informationen an weitere Seiten übermitteln. Jedoch ist dies auf den Einsatz von Formularen beschränkt. Eine globalere Lösung bieten Cookies und Sessions, die in diesem Kapitel vorgestellt werden.

8.1 Cookies

Cookies sind kleine Textinformationen, die vom Server an den Browser gesendet werden und die beim Surfer auf der Festplatte gespeichert werden. Bei späteren Zugriffen auf denselben Webserver sendet der Browser jeweils das Cookie wieder mit. Auf diese Art kann der Webserver einen Anwender »wiedererkennen«.

8.1.1 Cookies – allgemeine Eigenschaften

Es gibt dabei bestimmte Beschränkungen für Cookies, die jedoch in den einzelnen Browsern unterschiedlich implementiert sein können. Nach der Cookie-Spezifikation (*http://tools.ietf.org/html/rfc2965*) sollen Browser mindestens 300 Cookies speichern, die jeweils 4.096 Bytes groß sein dürfen und wobei 20 Cookies pro Domain erlaubt sind.

Cookies können dabei immer nur von der Domain gelesen werden, die das Cookie auch gesetzt hat. Umgangen wird diese Einschränkung jedoch manchmal dadurch, dass über Bilder, die von anderen Domains eingebunden werden, ebenfalls Cookies gesetzt werden.

Ob und welche Cookies gespeichert werden dürfen, bestimmt der Benutzer in seinen Browsereinstellungen. Im Firefox finden Sie die entsprechende Option unter EXTRAS/EINSTELLUNGEN/DATENSCHUTZ im Bereich COOKIES.

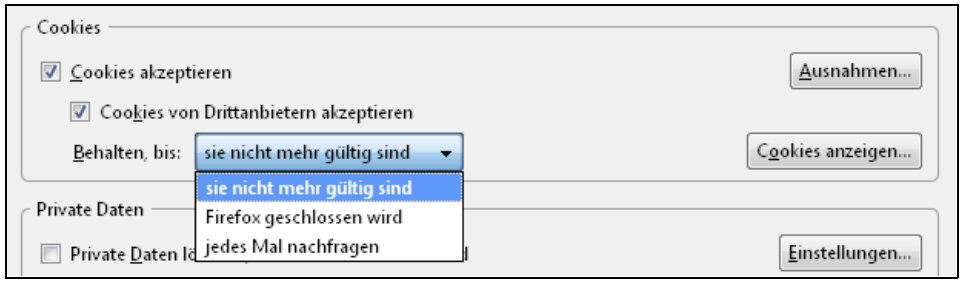


Abbildung 8.1: Einstellungen für Cookies im Firefox

Hier legen Sie fest, ob Sie allgemein Cookies akzeptieren möchten, ob Sie auch Cookies von Drittanbietern akzeptieren möchten und wie lange Sie die Cookies behalten möchten.

Tipp

Cookies von Drittanbietern sind Cookies, die über in einer Seite referenzierte Bilder auch von anderen Domains als der ursprünglichen Seite gesetzt werden können. Dadurch lassen sich allgemeine Profile des Surfverhaltens erstellen. Deswegen könnte es sinnvoll sein, diese nicht zuzulassen.

Über COOKIES ANZEIGEN verschaffen Sie sich einen Überblick über die auf Ihrem Computer gespeicherten Cookies.

Abbildung 8.2 zeigt beispielsweise, dass die Website *php.net* Cookies gesetzt hat: Das Cookie `LAST_LANG` speichert die zuletzt verwendete Sprache, im Beispiel `de`. Durch dieses Cookie kann bei der nächsten Benutzung die Funktionsreferenz des PHP-Manuals die Erläuterung gleich in der richtigen Sprache angezeigt werden.

8.1.2 Kommunikation zwischen Browser und Server

Cookies werden über den HTTP-Header versendet. Um das nachzuvollziehen, ist es nützlich, zu verstehen, wie eigentlich die Kommunikation zwischen Browser und Server funktioniert. Wenn Sie im Browser eine Adresse eingeben und absenden, so stellt der Browser eine Anfrage, die folgendermaßen beginnt:

```
GET /index.php HTTP/1.1
Host: www.maurice-web.de
```

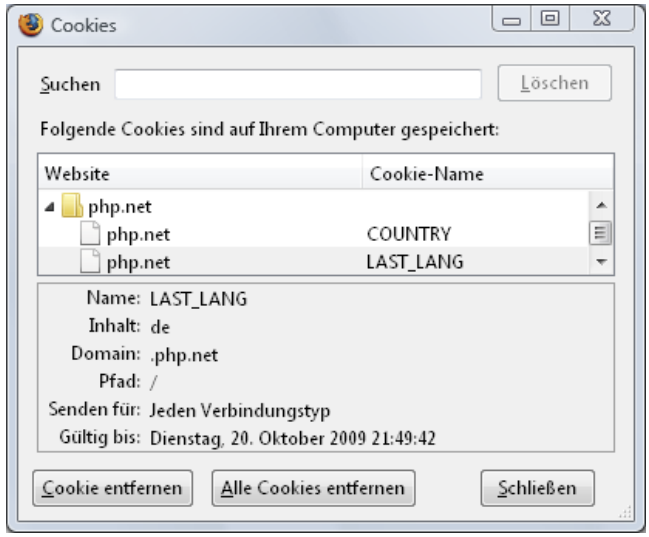


Abbildung 8.2: Die von der Website php.net gespeicherten Cookies

Zuerst steht die Übertragungsmethode, um Seiten anzufordern, GET ist die häufigste. Dann folgen der Name der angeforderten Datei und am Schluss die verwendete HTTP-Version. In der nächsten Zeile hinter `Host:` steht der eigentliche Domain-Name. Üblicherweise werden noch weitere Informationen mitgeschickt wie über den verwendeten Browser etc.

Der Server schickt in seinem Antwort-Header verschiedene Informationen. Am Anfang stehen das Protokoll und der Statuscode:

```
HTTP/1.1 200 OK
```

Darauf folgen weitere Informationen wie der MIME-Typ und die Länge des Dokuments, der verwendeter Server usw. Schließlich wird im Body der Antwort, wenn alles geklappt hat, das Dokument mitgesendet, das der Browser darstellt.

Noch einmal zurück zum Anfang: Cookies werden vom Server mit dem Header mitgesendet, nicht mit dem Body der Antwort, bei dem das Dokument gesendet wird. Auf diesen Punkt kommen wir etwas später noch einmal zu sprechen, wenn es darum geht, warum die Funktion zum Setzen von Cookies ganz am Anfang des Dokuments stehen muss.

8.1.3 Cookies setzen per PHP

Zum Setzen von Cookies stellt PHP die Funktion `setcookie()` zur Verfügung. Diese Funktion erlaubt mehrere Parameter.

```
setcookie("sprache", "en", time()+3600, "/", ".example.com", true, true);
```

Dabei bedeuten die einzelnen Parameter:

- der *Name* des Cookies, im Beispiel `sprache`
- der *Wert*, hier `en`
- das Verfallsdatum in Form eines Unix-Zeitstempels. Üblicherweise verwendet man hier den aktuellen Zeitstempel, zu dem man noch die gewünschte Anzahl an Sekunden addiert. Wenn Sie diesen Parameter nicht setzen, gilt das Cookie nur während dieser Browsersitzung und wird gelöscht, wenn jemand seinen Browser schließt.
- der *Pfad*. Standardmäßig gilt das Cookie nur für den Unterordner, in dem das Skript steht, das das Cookie gesetzt hat. Über diese Einstellung können Sie das ändern. Wenn Sie hier `/` angeben, ist das Cookie für die gesamte Domain gültig. Geben Sie hier hingegen `/manual/` an, ist das Cookie nur für den Ordner *manual* und alle Unterordner gültig.
- die *Domain*, für die das Cookie gilt. Das ist nur relevant, wenn Sie mit Subdomains arbeiten. Zwei mögliche Subdomains zu `example.com` könnten beispielsweise `de.example.com` und `en.example.com` sein. Normalerweise gilt das Cookie nur für die Subdomain, die das Cookie setzt. Soll ein Cookie hingegen in allen Subdomains von `example.com` gelten, schreiben Sie `.example.com`. Sie können hier aber, da Cookies immer nur von derselben Domain ausgelesen werden, nicht einfach eine fremde Domain angeben.
- Verbindungstyp: Wenn Sie für diesen Parameter `true` angeben, darf das Cookie nur über sichere HTTPS-Verbindungen versendet werden.
- `httponly` (ab PHP 5.2): Über den letzten Parameter können Sie durch die Angabe `true` festlegen, dass das Cookie nur über das HTTP-Protokoll ausgelesen wird und nicht beispielsweise per JavaScript. Das kann helfen, einen Identitätsklau über XSS zu verhindern, wird allerdings nicht in allen Browsern unterstützt.

Obligatorisch ist nur der erste Parameter – der Name – die anderen sind fakultativ.

8.1.4 Cookies setzen und auslesen

Die auf diese Art gesetzten Cookies können Sie über das von PHP bereitgestellte assoziative Array `$_COOKIES` auslesen. Die Name-Wert-Paare dieses Arrays entsprechen dem Namen und dem Wert der gesetzten Cookies.

Sehen wir uns das einmal am Beispiel an. Im ersten Skript werden zwei Cookies gesetzt. Außerdem gibt es einen Link auf eine weitere Datei, in der die Cookies ausgelesen werden. Dieses Mal ist das gesamte Listing inklusive (X)HTML-Teil angegeben, da der PHP-Code an verschiedenen Stellen steht.

Listing 8.1: Zwei Cookies werden gesetzt (cookie_setzen.php).

```
01 <?php
02 setcookie("name", "Marie", time()+7200);
03 setcookie("farbe", "rot", time()+7200);
04 ?>
05 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
06     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
07 <html xmlns="http://www.w3.org/1999/xhtml">
08  <head>
09   <title>Cookies setzen</title>
10   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
11 </head>
12 <body>
13 <p>In diesem Dokument sollten zwei Cookies gesetzt werden</p>
14 <p>Prüfen Sie, ob das geklappt hat über
15 <a href="cookie_auslesen.php">cookie_auslesen.php</p>
16 </body>
17 </html>
```

Im Beispiel wird in Zeile 2 ein Cookie gesetzt. Es erhält den Namen `name`, den Wert `Marie` und als Verfallsdatum wird `time()+7200` bestimmt. `time()` liefert den aktuellen Timestamp, zu dem 7200 Sekunden (60*60*2), also zwei Stunden, addiert werden. In Zeile 3 wird ein weiteres Cookie gesetzt, dieses Mal mit dem Namen `farbe` und dem Wert `rot`.

Achtung

Dass `setcookie()` ganz am Anfang steht, ist nicht Zufall: Da Cookies über den Header versendet werden, müssen Sie die Funktion `setcookie()` aufrufen, bevor irgendeine Ausgabe erfolgt.

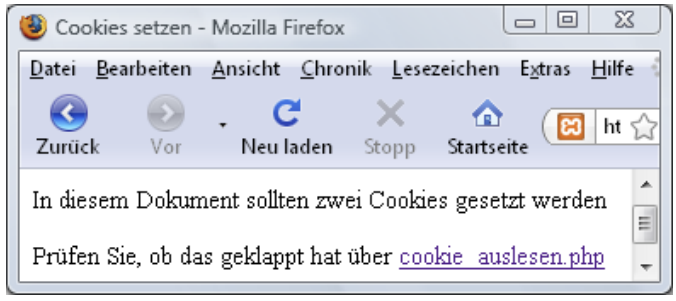


Abbildung 8.3: In diesem Dokument werden die Cookies gesetzt.

Das zweite Skript dient dazu, die Cookie-Werte auszulesen.

Listing 8.2: Die Werte werden wieder ausgelesen (cookie_auslesen.php).

```
if (isset($_COOKIE["name"]) && isset($_COOKIE["farbe"])) {
  echo "Cookies wurden gesetzt<br />\n";
  echo "Name: " . htmlspecialchars($_COOKIE["name"]) . "<br />\n";
  echo "Farbe: " . htmlspecialchars($_COOKIE["farbe"]);
} else {
  echo "keine Cookies gesetzt";
}
```

Im zweiten Skript wird geprüft, ob die Variablen `$_COOKIE["name"]` und gleichzeitig die Variable `$_COOKIE["farbe"]` existieren. Wenn das der Fall ist, werden die beiden Werte ausgegeben. Sind hingegen die Variablen nicht gesetzt, erscheint eine andere Meldung. Da das Setzen von Cookies ja im Browser deaktiviert werden kann, sollten Sie immer, bevor Sie auf die Werte zugreifen, testen, ob die entsprechenden Variablen gesetzt sind.

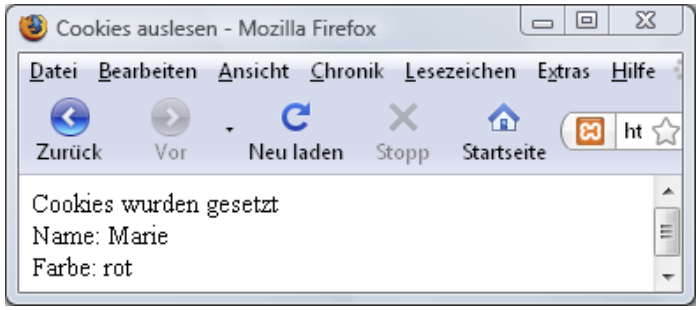


Abbildung 8.4: Die Werte werden ausgegeben.

Hinweis

Zum Löschen von Cookies gibt es keine eigene Funktion. Sie löschen ein Cookie, indem Sie ein Cookie mit denselben Parametern setzen, aber mit einer Ablaufzeit, die in der Vergangenheit liegt.

```
setcookie("name", "Marie", mktime(0, 0, 0, 1, 1, 1980));
```

8.1.5 Die einzelnen Schritte genau betrachtet

Sehen wir uns die einzelnen Schritte noch einmal im Detail an:

1. Das Dokument *cookie_setzen.php* wird vom Browser angefordert.

2. Der Server sendet das Dokument und versucht gleichzeitig das Cookie zu setzen. (Was fehlschlägt, wenn der Benutzer das Setzen von Cookies in seinen Browsereinstellungen unterbunden hat.)

3. Der Benutzer klickt auf den Link im Dokument *cookie_setzen.php* und das Dokument *cookie_auslesen.php* wird vom Browser angefordert. In seiner Anforderung sendet *der Browser gleichzeitig das Cookie* mit den Werten mit.

4. Der Server liefert das Dokument *cookie_auslesen.php* zurück, in dem der Inhalt des Cookies sichtbar ausgegeben wird.

5. Bei jeder weiteren Forderung von Seiten derselben Domain und im selben Unterordner sendet der Browser erneut das Cookie mit – so lange es noch gültig ist.

Tipp

Das lässt sich auch gut mit LiveHTTP-Headers verfolgen, einer kostenlosen Erweiterung für Firefox (*https://addons.mozilla.org/de/firefox/addon/3829*). Wenn Sie diese Erweiterung installiert haben, können Sie sich die Header in der Sidebar anzeigen lassen (ANSICHT/SIDEBAR/LIVEHTTP-HEADERS).

8.1.6 Headers already sent

Wenn Sie *irgendetwas vor dem Aufruf* von `setcookie()` *ausgeben lassen*, erhalten Sie die Warnung, dass die Header-Information nicht verändert werden kann, weil der Header bereits gesendet wurde. Außerdem werden die Cookies nicht gesetzt.

Abbildung 8.5: Warnung, wenn vor dem Aufruf von setcookie() etwas ausgegeben wird.

Wenn nämlich eine Ausgabe erfolgt ist, dann sendet PHP automatisch den für (X)HTML-Dokumente üblichen Header.

Hier ein paar Beispiele dafür, was alles als Ausgabe zählt:

- Zuerst einmal natürlich der (X)HTML-Code, unabhängig davon, ob Sie ihn direkt oder per PHP ausgeben lassen.
- Aber dazu zählen auch Leerzeichen, die vor dem öffnenden `<?php`-Tag stehen.
- Falls Sie Dateien per `include()` oder `require()` einbinden, zählen natürlich auch Ausgaben, die von diesen Dateien stammen!
- Leerzeichen und Zeilen innerhalb des PHP-Codes stören hingegen nicht, da sie keine Ausgabe erzeugen.
- Besonders tricky ist das Ganze bei der Verwendung von UTF-8 als Kodierung. Denn bei UTF-8-Dateien kann am Anfang ein so genanntes BOM integriert sein. BOM steht für Byte Order-Mark (Bytereihenfolge-Markierung) und enthält drei Bytes, die die Kodierung markieren. *PHP vor Version 5.3* interpretiert diese drei Bytes als Ausgabe und meldet entsprechend einen Fehler.

Tipp

Prinzipiell sollten Sie UTF 8 immer ohne BOM einsetzen. Wenn Sie den Editor Scite verwenden, können Sie das unter FILE/ENCODING einstellen; andere Editoren bieten entsprechende Einstellungen. Ansonsten lässt sich das BOM immer mit einem Hex-Editor entfernen.

8.1.7 Ausgabepufferung aktivieren

Um das Problem mit `Header already sent` zu umgehen, können Sie auch die Ausgabepufferung aktivieren. Dann werden Ausgaben erst in die Datei geschrieben, wenn Sie es bestimmen. Im PHP-Skript aktivieren Sie die Ausgabepufferung über die Funktion `ob_start()`. Alles, was Sie danach per `echo` oder über eine andere Funktion ausgeben, wird in den Puffer geschrieben. Erst wenn Sie `ob_end_flush()` angeben, findet die Ausgabe wirklich statt. PHP kümmert sich dann selbst darum, dass der Befehl zum Setzen von Cookies im Header steht.

Listing 8.3: Ist die Ausgabepufferung aktiviert, kann der Befehl zum Setzen von Cookies auch später im Skript erfolgen (cookie_ausgabepufferung.php).

```
01 <?php
02 ob_start();
03 ?>
04 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
05     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
06 <html xmlns="http://www.w3.org/1999/xhtml">
07  <head>
```

```
08   <title>Cookie setzen mit Ausgabepufferung</title>
09   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
10 </head>
11 <body>
12 <?php
13 setcookie("stadt", "München", time()+3600);
14 echo "Cookies werden mitten im Skript gesetzt und es funktioniert!";
15 ob_end_flush();
16 ?>
17 </body>
18 </html>
```

8.1.8 Cookies und Sicherheit

Können Sie den Inhalten von Cookies vertrauen? Leider nein. Cookies können beliebig manipuliert werden. Schließlich ist es eine Textdatei, die auf der Festplatte des Surfers gespeichert ist. Besonders komfortabel kann ein Benutzer Cookies etwa über die Webdeveloper Toolbar (*http://www.erweiterungen.de/detail/Web_Developer/*) erstellen, die Firefox-Erweiterung, die Sie auch schon in Kapitel 7 kennen gelernt haben. Hierfür braucht er nur in der Webdeveloper-Symbolleiste COOKIES/COOKIE HINZUFÜGEN zu wählen. Das bedeutet, Sie müssen die Inhalte von Cookies genauso prüfen, wie Sie die von Formularen gesendeten Daten prüfen müssen (siehe hierzu Kapitel 7).

8.2 Sessions – Sitzungen

Auch Sessions werden dazu verwendet, Benutzer wiederzuerkennen und Daten über mehrere Webseiten hinweg verfügbar zu halten. Der grundlegende Unterschied zu Cookies ist aber der, dass die Daten bei Sessions nicht auf dem Client (Browser), sondern *auf dem Server* gespeichert werden. Auf dem Client wird hingegen nur ein Cookie gespeichert, das den Client beim Server identifiziert. Ein weiterer Unterschied ist, dass Sie über Sessions üblicherweise nur Informationen während einer Browsersitzung speichern. Eine Sitzung geht so lange, bis der Browser geschlossen wird.

Um eine Session zu starten, verwenden Sie den PHP-Befehl `session_start()`. Wenn Sie eine Session gestartet haben, können Sie das von PHP bereitgestellte vordefinierte assoziative Array `$_SESSION` verwenden, um Schlüssel-Wert-Paare zu speichern.

Im ersten Skript werden Informationen in `$_SESSION` abgelegt.

Listing 8.4: Eine Session starten und Werte speichern (session.php)

```
01 <?php
02 session_start();
03 ?>
04 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
05     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
06 <html xmlns="http://www.w3.org/1999/xhtml">
07  <head>
08   <title>Session verwenden</title>
09   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
10 </head>
11 <body>
12 <?php
13 $_SESSION["name"]  = "Marie";
14 $_SESSION["farbe"] = "rot";
15 echo "Dieses Mal werden Werte über Sessions gesetzt. ";
16 echo "Hier können Sie sie auslesen <a
     href='session_auslesen.php'>session_auslesen.php
17 </a>";
18 ?>
19 </body>
20 </html>
```

In Zeile 2 wird der Befehl zum Starten einer Session aufgerufen. Dieser muss – genau wie `setcookie()` – stehen, bevor irgendeine Ausgabe erfolgt ist. In Zeile 13 und 14 wird das `$_SESSION`-Array zur Speicherung der Werte benutzt.

Abbildung 8.6: Die Ausgabe von session.php

Klickt der Nutzer auf den bereitgestellten Link, werden im Skript *session_auslesen.php* die Werte ausgegeben.

Listing 8.5: Die über die Session gespeicherten Informationen werden wieder ausgelesen (session_auslesen.php).

```
01 <?php
02 session_start();
03 ?>
04 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
05     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
06 <html xmlns="http://www.w3.org/1999/xhtml">
07  <head>
08   <title>Session verwenden</title>
09   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
10 </head>
11 <body>
```

```
12 <?php
13 if (isset($_SESSION["name"]) && isset($_SESSION["farbe"])) {
14   echo "Die geschriebenen Werte sind: <br />";
15   echo "Name: {$_SESSION['name']} <br />\n";
16   echo "Farbe: {$_SESSION['farbe']} <br />\n";
17 } else {
18   echo "Noch keine Session gesetzt";
19 }
20
21 ?>
22 </body>
23 </html>
```

Um auf die in der Session gespeicherten Werte zugreifen zu können, muss zuerst wieder `session_start()` in Zeile 2 aufgerufen werden. Mit `if` wird in Zeile 13 geprüft, ob die entsprechenden Variablen gesetzt sind, und diese werden dann ausgegeben.

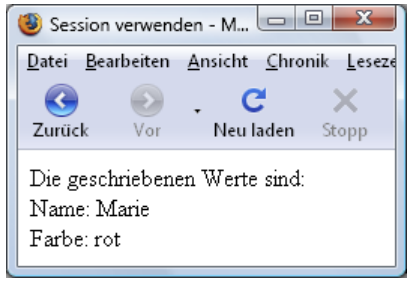


Abbildung 8.7: Die Werte werden ausgegeben.

8.2.1 Speicherung von Session-Informationen

Sehen wir uns einmal an, wo diese Informationen gespeichert werden. Im Browser findet sich ein Cookie mit dem Namen `PHPSESSID` – und mit einem ziemlich kryptischen Wert, den PHP selbst vergibt (Abbildung 8.8).

Die eigentlichen Informationen, die gespeichert werden, verbleiben auf dem Server. Bei der XAMPP-Installation unter Windows ist das Verzeichnis, in dem diese Daten gespeichert werden, das Unterverzeichnis *tmp* innerhalb von *xampp*. Hier finden Sie eine Datei, die mit `sess_` beginnt, und darauf folgt der als Inhalt des Cookies vergebene Wert. Darin stehen die gespeicherten Daten in serialisierter Form.

8.2.2 Sessions bei deaktivierten Cookies

Da die Identifizierung des Clients über ein Cookie geschieht, funktionieren Sessions bei vollständiger Deaktivierung von Cookies durch den Benutzer nicht. Er muss zumindest temporäre Cookies zulassen, d. h. diejenigen, die bis zum Beenden des Browsers erhalten bleiben.

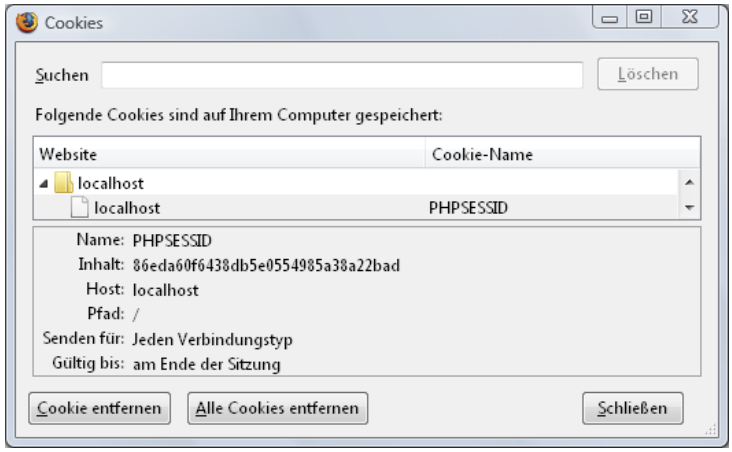


Abbildung 8.8: Auf dem Client wird ein Cookie mit der Session-ID gespeichert.

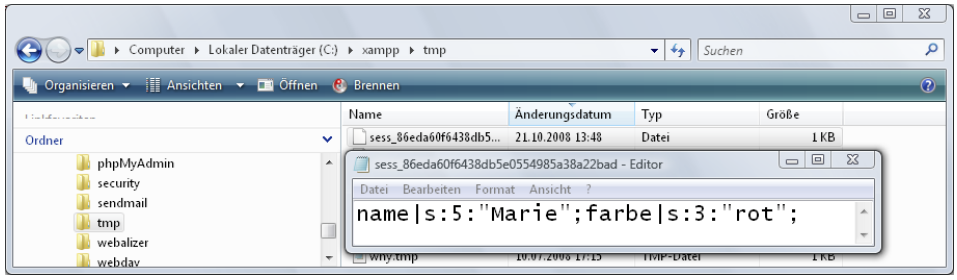


Abbildung 8.9: Im entsprechenden Unterordner auf dem Webserver sind die Sessiondaten gespeichert.

Sollen Sessions auch bei deaktivierten Cookies funktionieren, müssen Sie die Session-ID über die URL übergeben. In unserem Beispiel muss man dafür die Datei *session.php* bearbeiten, an der Stelle, an der der Link steht. Diesen muss man um die Session-Information erweitern. Dafür brauchen Sie den von PHP vergebenen Sessionnamen, den `session_name()` liefert, und den Wert, den Sie über `session_id()` ermitteln können.

Listing 8.6: An den Link wird die Session-Information angehängt (session_id_url.php).

```
echo "Hier können Sie sie auslesen <a href='session_auslesen.php?"
     . session_name()  . "="
     . session_id()
     . "'>session_auslesen.php</a>";
```

Abbildung 8.10: Hier wird die Session-ID über die URL übertragen.

8.3 Ein Login-System mit Sessions

Eine klassische Anwendung für Sessions ist ein Login-System, das nur berechtigten Benutzern Zugriff auf bestimmte Informationen erlaubt. Wie so etwas geht, soll nun gezeigt werden.

Das Beispiel besteht aus mehreren Dateien:

- *start.php* beinhaltet die Startseite mit dem Formular, in das der Benutzer seinen Benutzernamen und sein Passwort eintragen kann.
- In *login.php* werden die Login-Informationen verarbeitet. Wenn die eingegebenen Daten korrekt sind, findet eine Weiterleitung zur Datei *willkommen.php* statt. Sind die Daten hingegen nicht korrekt, wird wieder zur Startseite umgeleitet und es erscheint eine Fehlermeldung.
- Die Seite *willkommen.php* erreicht nur ein dazu berechtigter Benutzer. Er findet hier auch einen Link auf *logout.php*, um sich auszuloggen. Wird die Seite *willkommen.php* ohne Berechtigung aufgerufen, wird der Benutzer wieder auf *start.php* umgeleitet.
- In *logout.php* werden alle Session-Informationen gelöscht, und der Benutzer wird wieder zur Startseite umgeleitet.

Jetzt zum ersten Skript: *start.php*.

Listing 8.7: Die Startseite mit dem Login-Formular (start.php)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Login mit Sessions</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<style type="text/css">
  .fehler { color: red; }
</style>
<body>
<?php
if (isset($_GET["f"]) && $_GET["f"] == 1) {
  echo "<p class='fehler'>Login-Daten nicht korrekt</p>";
}
?>
<form method="post" action="login.php">
Ihr Name: <br />
<input type="text" name="name" size="20" />
<br />
Passwort: <br />
<input type="password" name="passwort" size="20" /><br />
```

```
23 <input type="submit" value="Login" />
24 </body>
25 </html>
```

start.php beinhaltet ab Zeile 17 das Login-Formular. Die Daten werden an das Skript *login.php* gesendet.

Tipp

Die Formulardaten und damit auch das Passwort werden im Beispiel über HTTP versendet. Die ganze Kommunikation per HTTP kann jedoch »mitgehört« werden. Deswegen sollten Sie in solchen Fällen die Verbindung über HTTPS absichern. Mehr Informationen dazu unter *http://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure*.

Erklärungsbedürftig ist auf jeden Fall die Überprüfung in Zeile 13: Hier wird, wenn `$_GET["f"]` den Wert 1 hat, eine zusätzliche Fehlermeldung ausgegeben. Dieser Parameter wird in *login.php* gesetzt, wenn sich jemand mit falschen Daten einloggt.

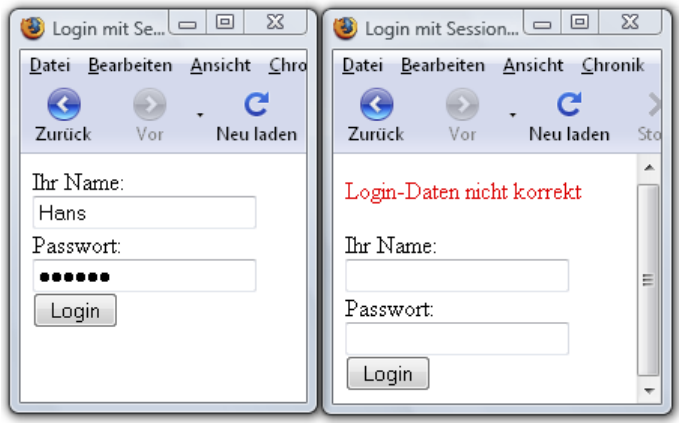


Abbildung 8.11: Links das Login-Formular beim ersten Mal, rechts das Login-Formular bei einem Fehler

login.php beinhaltet die Verarbeitung der im Formular eingegebenen Daten. Sehen wir uns hier erst einmal die etwas vereinfachte Version an:

Listing 8.8: Die Überprüfung der Benutzerdaten in vereinfachter Version (login_einfach.php)

```
01 <?php
02 session_start();
03 if (isset($_POST["name"])
```

```
04    && $_POST["name"] == "Hans"
05    && $_POST["passwort"] == "geheim") {
06   $_SESSION["name"] = "Hans";
07   $_SESSION["login"] = "ok";
08   header("Location: willkommen.php");
09 } else {
10   header("Location: start.php?f=1");
11 }
12 ?>
```

In Zeile 2 wird eine Session gestartet. Dann folgt (Zeile 3) eine Überprüfung, ob das Formular abgesendet und Name und Passwort die Werte »Hans« und »geheim« haben. Bei einer echten Anwendung würden diese Daten wahrscheinlich nicht hartkodiert im Skript stehen, sondern aus einer Datenbank stammen.

Wenn die Daten korrekt sind, werden zwei Session-Variablen erstellt, eine mit dem Benutzernamen und eine weitere mit einer Kennung, dass das Login korrekt war. Außerdem findet eine Umleitung auf die Seite *willkommen.php* statt.

Hinweis

Die Umleitung wird über die Funktion `header()` realisiert, die allgemein zum Senden von HTTP-Headern dient. Deswegen muss diese Funktion – genau wie `session_start()` oder `setcookie()` eingesetzt werden, bevor eine Ausgabe erfolgt ist. Für die Umleitung schreiben Sie `Location:` und den gewünschten Pfad. Weitere `header()` können mehr als Umleitungen, Genaueres lesen Sie im Manual unter *http://www.php.net/manual/de/function.header.php*.

Sind die eingegebenen Daten hingegen nicht korrekt, wird der Benutzer zurück auf die *start.php*-Seite geführt, zusätzlich wird (Zeile 10) noch ein Parameter angehängt, nämlich `f=1`. Auf diesen wird im Skript *start.php* in der Zeile 13 zurückgegriffen, um eine Fehlermeldung auszugeben.

Zwei Punkte sollen an diesem Skript noch verbessert werden: Zuerst einmal funktioniert das Skript derzeit nicht, wenn jemand Cookies vollständig deaktiviert hat. Abhilfe ist einerseits über die Einstellung `session.use_trans_sid` möglich, dann hängt PHP bei allen Links automatisch Sessionname und Session-ID an. Praktische Alternative hier: Sie erledigen es in diesem Fall selbst. Mit `session_name()` können Sie auf den Namen der Session zugreifen und mit `session_id()` auf den Wert. Diese Angaben werden an die URL drangehängt.

```
"willkommen.php?" . session_name() . "=" . session_id();
```

Zweiter Punkt, der korrigiert werden soll. Die Umleitung wird ja über `Location` realisiert. Das Protokoll HTTP 1.1 verlangt eine absolute URL bei `Location` (*http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.30*), auch wenn viele Clients hier mit relativen URLs zurechtkommen. Die absolute URL könnten Sie natürlich direkt ange-

ben – das hat aber den Nachteil, dass Sie diese Stelle dann anpassen müssen, wenn das Skript woanders laufen soll. Deswegen lassen wir die absolute URL automatisch von PHP ermitteln.

Mehrere Bestandteile werden für den absoluten Pfad zusammengefügt: `$host` beinhaltet die eigentliche Domain, `$uri` den Unterordner und der eigentliche Dateiname mit eventuell vorhandenen Parametern wird in `$extra` gespeichert.

Listing 8.9: Die Überprüfung, ob Benutzername und Passwort korrekt sind (login.php)

```
<?php
session_start();
$host = htmlspecialchars($_SERVER["HTTP_HOST"]);
$uri  = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
if (isset($_POST["name"])
   && $_POST["name"] == "Hans"
   && $_POST["passwort"] == "geheim") {
  $_SESSION["name"] = "Hans";
  $_SESSION["login"] = "ok";
  $extra = "willkommen.php?" . session_name() . "=" . session_id();
} else {
  $extra = "start.php?f=1";
}
header("Location: http://$host$uri/$extra");
?>
```

Fehlt schließlich noch die Datei *willkommen.php*, die die Seite mit den zu schützenden Informationen enthält. Hier muss natürlich sichergestellt werden, dass diese Seite nicht ohne vorheriges korrektes Login aufgerufen wird.

Listing 8.10: Die Willkommensseite nur für berechtigte Benutzer (willkommen.php)

```
<?php
session_start();
if (isset($_SESSION["login"]) && $_SESSION["login"] == "ok") {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Willkommen im geschützten Bereich</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<?php
  echo "<h1>Hallo {$_SESSION['name']}</h1>";
?>
<p>Hier stehen viele weitere interessante Informationen</p>
```

```
17 <p><a href="logout.php">Ausloggen</p>
18 </body>
19 </html>
20 <?php
21 } else {
22   $host  = htmlspecialchars($_SERVER["HTTP_HOST"]);
23   $uri   = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
24   $extra = "start.php";
25   header("Location: http://$host$uri/$extra");
26 }
27 ?>
```

In Zeile 2 wird über `session_start()` die Session gestartet. In Zeile 3 findet die Überprüfung statt, ob die betreffende Person eingeloggt ist, das ist sie, wenn `$_SESSION["login"]` den Wert `ok` hat. In Zeile 14 wird dann etwas Text ausgegeben. Falls jedoch jemand nicht korrekt eingeloggt ist, wird er über den `else`-Zweig, der in Zeile 21 beginnt, wieder auf die Startseite umgeleitet.

Abbildung 8.12: Die Willkommens-Seite ist geschützt.

Bei Klick auf den Link AUSLOGGEN im *willkommen.php*-Skript wird *logout.php* aufgerufen. Dieses Skript sorgt dafür, dass die Sessioninformationen und das Cookie gelöscht werden und der Benutzer wieder zur Startseite gelangt.

Listing 8.11: Das Skript zum Ausloggen (logout.php)

```
01 <?php
02 session_start();
03 $_SESSION = array();
04 if (isset($_COOKIE[session_name()])) {
05     setcookie(session_name(), "", time()-42000, "/");
06 }
07 session_destroy();
08 $host  = htmlspecialchars($_SERVER["HTTP_HOST"]);
09 $uri   = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
```

```
10 $extra = "start.php";
11 header("Location: http://$host$uri/$extra");
12 ?>
```

In Zeile 2 wird erst einmal noch die Session gestartet. In Zeile 3 wird das `$_SESSION`-Array auf ein leeres Array gesetzt, damit werden also alle Inhalte gelöscht. Außerdem wird in Zeile 4 das Cookie gelöscht – sofern eines gesetzt wurde. Wenn eines gesetzt wurde, dann existiert im `$_COOKIE`-Array die Variable mit dem Sessionname als Schlüssel. In diesem Fall (Zeile 5) wird das Cookie gelöscht, indem das Ablaufdatum auf einen negativen Wert gesetzt wird. Schließlich wird die Sitzung über `session_destroy()` zerstört. Für die Umleitung auf die Startseite wird die absolute URL wieder aus den einzelnen Bestandteilen zusammengesetzt.

Passwörter besser abspeichern

Im Beispiel werden die Passwörter im Klartext im Skript *login.php* gespeichert. Auch wenn man sich vorstellt, dass man später die Benutzerdaten – Benutzername und Passwort – in einer Datenbank speichert, ist die Speicherung im Klartext nicht optimal. Wenn Passwörter im Klartext gespeichert sind und jemand erhält aus irgendeinem Grund Zugang zu der Passwortliste, kann er sich als beliebiger Benutzer einloggen. Besser ist es, das Passwort mit einem MD5-Hash zu schützen.

Hierfür gibt es in PHP die Funktion `md5()`. Diese transformiert Strings nach einem bestimmten Algorithmus, wobei der Prozess nicht umkehrbar ist: Sie können einen mit `md5()`-transformierten String nicht wieder in die ursprüngliche Form umwandeln. Das aber genau ist der Vorteil: Falls jemand den transformierten Passwort-String sieht, kennt er nicht das Klartext-Passwort.

Sie speichern also das mit der `md5()`-Funktion bearbeitete Passwort (in der Datenbank oder sonst wo). Gibt jemand sein Passwort ein, verwenden Sie wieder die `md5()`-Funktion und vergleichen das Ergebnis mit dem gespeicherten, ebenfalls per `md5()` transformierten String. In unserem Beispiel könnte das folgendermaßen aussehen (im Listing sind die Zeilen aus *login.php* aufgeführt, in denen es Änderungen gibt):

Listing 8.12: Einsatz der md5()-Funktion beim Login (login_md5.php)

```
05 if (isset($_POST["name"])
06    && $_POST["name"] == "Hans"
07    && md5($_POST["passwort"]) == "e8636ea013e682faf61f56ce1cb1ab5c")
```

`e8636ea013e682faf61f56ce1cb1ab5c` ist dabei der über ein

```
echo md5("geheim");
```

gewonnene String.

Damit das aber sicher ist, sollte man natürlich bessere Passwörter als »geheim« benutzen, d. h. längere Kombinationen aus Zahlen und Buchstaben in Groß- und Kleinschreibung.

Apropos Sicherheit: Wie verhält es sich eigentlich allgemein bzgl. der Sicherheit bei Sessions? An sich sind die Daten sicherer als bei Cookies, da sie ja beim Server und nicht im Client gespeichert werden. Falls jemand aber eine Session-ID »errät« und ein Cookie mit der Session-ID an den Server liefert, wird dieser ihn für die andere Person halten. Und an sich ist es natürlich auf jeden Fall sicherer, wenn die Session-ID nicht über die URL übertragen, sondern in einem Cookie gespeichert wird. Die Hauptgefahr besteht also im *Session-Hijacking*, das heißt, dass jemand auf einer fremden Session surft. Möglichkeiten, das zu verhindern, sind unter *http://shiflett.org/articles/the-truth-about-sessions* beschrieben.

9 Objektorientierung

An verschiedenen Stellen sind Sie in den vorherigen Kapiteln bereits mit der Objektorientierung in Berührung gekommen. Dieses Kapitel fasst das bisher Gesagte zusammen und führt Sie in fortgeschrittene Techniken bei der Objektorientierung ein.

9.1 Methoden und Eigenschaften

Sie erinnern sich: In der objektorientierten Herangehensweise ist alles ein Objekt. Und Objekte haben verschiedene *Eigenschaften* (Variablen) und *Methoden* (Funktionen). Die Baupläne der Objekte sind die Klassen. Sie definieren, welche Eigenschaften und Methoden vorgesehen sind und bei den konkreten Objekten werden diese dann gefüllt bzw. ausgeführt.

Hinweis

In PHP konnten Sie auch schon in Version 4 objektorientiert programmieren. Hier fehlen jedoch wichtige Features der Objektorientierung, die erst in PHP 5 implementiert sind – wie zum Beispiel die Möglichkeit, den Zugriff auf Methoden und Eigenschaften zu beschränken.

Wir werden im Folgenden mehrmals mit einer Klasse `Kunde` arbeiten, die Eigenschaften von Kunden speichert. Die allererste einfache Version kennen Sie schon aus Kapitel 5, die jetzt im Folgenden erweitert werden soll:

Listing 9.1: Ein Objekt der Klasse Kunde wird erstellt (kunde_beispiel.php).

```
01 class Kunde
02 {
03   public $name;
04   public function halloSagen()
05   {
06     echo "Hallo";
07   }
08 }
09 $neuerKunde = new Kunde();
10 $neuerKunde->name = "Anja";
```

```
11 $neuerKunde->halloSagen();
12 echo " ";
13 echo $neuerKunde->name;
```

Hier wird eine Klasse `Kunde` mit einer Eigenschaft – `$name` – und einer Methode – `halloSagen()` erstellt. In Zeile 9 wird ein neuer Kunde über das Schlüsselwort `new` erstellt. Diesem Kunden wird ein Name zugewiesen – die Eigenschaft `$name` gesetzt – und es wird die Methode `halloSagen()` aufgerufen. Zum Zugriff auf Eigenschaften und Methoden dient `->`. Das Skript gibt »Hallo Anja« aus.

9.2 Konstruktor und Destruktor

Ein Kunde ohne Namen wäre nicht so recht sinnvoll. Wenn Sie sichergehen möchten, dass immer, wenn ein neuer Kunde erstellt wird, auch sein Name angegeben wird, können Sie einen Konstruktor einsetzen.

Ein Konstruktor ist eine Methode mit dem vorgegebenen Namen `__construct()`. Die Konstruktormethode wird automatisch aufgerufen, wenn ein neues Objekt erstellt wird.

Achtung

Am Anfang dieses Methodennamens stehen wirklich zwei Unterstriche.

Erweitern wir das Beispiel durch eine Konstruktormethode:

```
01 class Kunde
02 {
03   public $name;
04   public function __construct($name)
05   {
06     $this->name = $name;
07   }
08   public function halloSagen()
09   {
10     echo "Hallo {$this->name}";
11   }
12 }
```

Dem Konstruktor wird in runden Klammern die Variable übergeben, im Rumpf des Konstruktors wird der übergebene Wert der Eigenschaft `name` zugewiesen. Dies geschieht über `$this->name`. Das Schlüsselwort `$this` enthält eine Referenz auf das *aktuelle Objekt* – das bei der Definition der Klasse ja noch nicht bekannt ist. Auch in der Methode `halloSagen()` wird über `$this->name` auf den übergebenen Namen zugegriffen.

Um ein Objekt dieser veränderten Klasse zu initialisieren, müssen Sie nun den Namen übergeben.

Listing 9.2: Die Klasse Kunde mit Konstruktor (konstruktor.php)

```
13 $neuerKunde = new Kunde("Anja");
14 $neuerKunde->halloSagen();
15 echo $neuerKunde->name;
```

Diese Zeilen geben bei Verwendung der angepassten Klasse wieder »Hallo Anja« aus.

Parallel zum Konstruktor gibt es auch einen Destruktor mit Namen `__destruct()`. Diese Methode wird automatisch aufgerufen, wenn ein Objekt zerstört wird und ist praktisch für »Aufräumarbeiten«, die noch durchgeführt werden sollen.

Listing 9.3: Beispiel für einen Destruktor (destruktor.php)

```
01 class Beispielklasse
02 {
03   public function __construct()
04   {
05     echo get_class($this) . " wurde erstellt<br />";
06   }
07   public function __destruct()
08   {
09     echo get_class($this) . " wurde zerstört";
10   }
11 }
12 $objekt = new Beispielklasse();
13 unset($objekt);
```

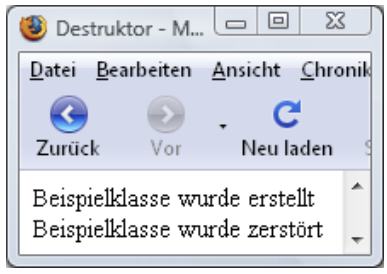


Abbildung 9.1: Konstruktor und Destruktor werden automatisch aufgerufen.

Im Listing wird eine Klasse `Beispielklasse` erstellt, die nur zwei Methoden hat: eine `__construct()`- und eine `__destruct()`-Methode. In diesen wird jeweils die aktuelle Klasse, die mit `get_class($this)` ermittelt wird, sowie ein passender Text ausgegeben.

Danach wird ein neues Objekt der Beispielklasse erstellt (Zeile 12) und über die Funktion `unset()` wieder zerstört. Beide definierten Methoden werden automatisch aufgerufen: `__construct()` bei Erstellung des Objekt, `__destruct()` bei der Vernichtung. Dieselbe Ausgabe haben Sie auch ohne expliziten Aufruf von `unset()`, da Objekte am Ende eines Skripts automatisch zerstört werden.

9.3 Objekte verschachteln

Objekte können auch verschachtelt werden. Im folgenden Beispiel gibt es eine Klasse `BeispielA`:

```
class BeispielA
{
  public function ausgabe()
  {
    echo "Ausgabe aus BeispielA";
  }
}
```

Außerdem wird eine Klasse `BeispielB` definiert. In ihrem Konstruktor wird eine neue Instanz der Klasse `BeispielA` erstellt:

```
class BeispielB
{
  public $a;
  public function __construct()
  {
    $this->a = new BeispielA();
  }
}
```

Wenn man jetzt eine Instanz der Klasse `BeispielB` erzeugt, kann man auf die dort erstellte Instanz von `BeispielA` zugreifen und deren Methode `ausgabe()` aufrufen.

Listing 9.4: Verschachtelte Objekte (obj_verschachteln.php)

```
$b = new BeispielB();
$b->a->ausgabe();
```

Ausgegeben wird entsprechend »Ausgabe aus BeispielA«. Mit solchen verschachtelten Objekten kommen Sie wieder in Kapitel 12 in Berührung.

9.4 Konstanten definieren

Im Beispiel wurden Variablen für Eigenschaften verwendet. Sie können auch Konstanten benutzen, deren Wert unveränderlich ist. Diese definieren Sie in der Klasse mit dem Schlüsselwort `const`. Sie können nur lesend auf diese Konstanten zugreifen. Das nächste Beispiel zeigt eine Beispielklasse mit einer Konstanten und den Zugriff auf diese – innerhalb und außerhalb der Klassendefinition.

Listing 9.5: Konstanten definieren (konstanten.php)

```
01 class Klasse {
02   const KONST_WERT = 42;
03   public function ausgabe()
04   {
05     echo self::KONST_WERT;
06   }
07 }
08 echo Klasse::KONST_WERT;
09 echo "<br />\n";
10 $obj = new Klasse();
11 $obj->ausgabe();
```

Zum Zugriff auf die Konstante verwenden Sie innerhalb der Klasse `self::KONST_WERT` (Zeile 5). Außerhalb der Klasse können Sie den Wert einer Konstanten über den Klassennamen gefolgt von zwei Doppelpunkten und dem Namen der Konstanten ausgeben: `Klasse::KONST_WERT` (Zeile 8). Im Beispiel wird zweimal 42 ausgegeben.

Zum Zugriff auf Konstanten dient der Doppel-Doppelpunkt-Operator, der auch Gültigkeitsbereichsoperator oder *Paamayim Nekudotayim* genannt wird. *Paamayim Nekudotayim* heißt Doppel-Doppelpunkt auf Hebräisch. Sie werden noch weitere Einsatzbereiche für diesen Operator kennen lernen.

9.5 Mehr Funktionalität bei der Klasse Kunde

Um weitere wichtige Konzepte der Objektorientierung zu demonstrieren, kehren wir noch einmal zu unserer Beispielklasse Kunde zurück und implementieren etwas mehr Funktionalität: Die Kunden sollen Speicherplatz zur Verfügung gestellt bekommen und über die Klasse soll die Speichermenge verwaltet werden. Die Speichermenge ist selbstverständlich begrenzt.

Dafür soll es drei zusätzliche Methoden geben:

- Die Methode `speichern()` überprüft zuerst beim Speichern, ob genügend Speicher zur Verfügung steht, und verringert den freien Speicher nach dem Speichervorgang.

- Die Methode `speicherFreigeben()` dient zum Freigeben von Speicher.
- Außerdem gibt es eine Methode namens `zustandAusgeben()`, die den aktuell verfügbaren Speicher ausliest.

Am Anfang der Klassendefinition werden die benötigten Eigenschaften aufgeführt.

```
01 class Kunde
02 {
03   public $name;
04   public $speicherGesamt = 50;
05   public $speicherVerbraucht;
```

Der Gesamtspeicher wird auf 50 gesetzt. Eine weitere Eigenschaft steht für den verbrauchten Speicher.

Es ändert sich auch der Konstruktor, d. h. die Methode, die beim Erstellen der Klasse automatisch aufgerufen wird. Sie erwartet nun neben dem Namen auch den bereits verbrauchten Speicher. Wenn der zweite Parameter nicht angegeben wird, wird von einem Speicherverbrauch von 0 ausgegangen.

Tipp

Genauso wie bei Funktionen können Sie bei Methoden Defaultwerte für Parameter vergeben.

```
07   public function __construct($name, $speicherVerbraucht = 0)
08   {
09     $this->name = $name;
10     $this->speicherVerbraucht = $speicherVerbraucht;
11   }
```

Außerdem werden im Konstruktor beide Parameter den Eigenschaften zugewiesen.

Die Methode `halloSagen()` bleibt gleich:

```
13   public function halloSagen()
14   {
15      echo "Hallo {$this->name}";
16   }
```

Nun kommt die neue Methode `speichern()`. Ihr wird der gewünschte Speicherbedarf als Parameter übergeben:

```
18   public function speichern($speicherBedarf)
19   {
20     if (($this->speicherGesamt - $this->speicherVerbraucht) >=
```

```
21          $speicherBedarf) {
22      $this->speicherVerbraucht =
23           $this->speicherVerbraucht+$speicherBedarf;
24      echo "$speicherBedarf gespeichert";
25    } else {
26      echo "$speicherBedarf nicht gespeichert. Nicht genügend Speicher mehr
    frei.";
27    }
28  }
```

Zuerst wird überprüft, ob genügend Speicher vorhanden ist (Zeile 20–21). Dafür muss der Gesamtspeicher minus dem verbrauchten Speicher größer oder gleich sein als der Speicherbedarf. Dann wird der aktuell verbrauchte Speicher gespeichert und außerdem eine Meldung ausgegeben. Steht nicht genügend Speicher zur Verfügung, wird eine entsprechende Meldung ausgegeben.

Die Methode zur Freigabe von Speicher ist schnell implementiert: Sie erwartet als Parameter die freizugebende Speichermenge. Im Methodenrumpf wird die verbrauchte Menge an Speicherplatz aktualisiert und eine Meldung ausgegeben.

```
29  public function speicherFreigeben($speicher)
30  {
31    $this->speicherVerbraucht = $this->speicherVerbraucht -
32      $speicher;
33    echo "$speicher Speicher frei gegeben";
34  }
```

Fehlt nur noch die Methode, die den aktuell verbrauchten und den aktuell zur Verfügung stehenden Speicher ausgibt:

```
35  public function zustandAusgeben()
36  {
37    $speicherFrei = $this->speicherGesamt -
38      $this->speicherVerbraucht;
39    echo "<p>Derzeit sind {$this->speicherVerbraucht}
40      Speicher verbraucht<br />";
41    echo "Es sind damit noch $speicherFrei frei</p>";
42  }
```

So sieht der Code der Klasse im Gesamtzusammenhang aus:

```
01 class Kunde
02 {
03  public $name;
04  public $speicherGesamt= 50;
05  public $speicherVerbraucht;
06
07  public function __construct($name, $speicherVerbraucht=0)
08  {
```

```
09     $this->name = $name;
10     $this->speicherVerbraucht = $speicherVerbraucht;
11   }
12
13   public function halloSagen()
14   {
15    echo "Hallo {$this->name}";
16   }
17
18   public function speichern($speicherBedarf)
19   {
20     if (($this->speicherGesamt - $this->speicherVerbraucht)
21         >= $speicherBedarf) {
22        $this->speicherVerbraucht =
23         $this->speicherVerbraucht+$speicherBedarf;
24        echo "$speicherBedarf gespeichert";
25     } else {
26        echo "$speicherBedarf Nicht gespeichert. Nicht genügend Speicher mehr
     frei.";
27     }
28   }
29   public function speicherFreigeben($speicher)
30   {
31     $this->speicherVerbraucht= $this->speicherVerbraucht -
32     $speicher;
33     echo "$speicher Speicher frei gegeben";
34   }
35   public function zustandAusgeben()
36   {
37    $speicherFrei = $this->speicherGesamt -
38    $this->speicherVerbraucht;
39     echo "<p>Derzeit sind {$this->speicherVerbraucht} Speicher verbraucht<br />";
40     echo "Es sind damit noch $speicherFrei frei</p>\n";
41   }
42 }
```

Jetzt kann man einen neuen Kunden erstellen und die Methoden aufrufen:

Listing 9.6: Die Klasse Kunde wurde um weitere Methoden ergänzt (kunde_erweitert.php).

```
43 $neuerKunde = new Kunde ("Anja");
44 $neuerKunde->halloSagen();
45 $neuerKunde->zustandAusgeben();
46 $neuerKunde->speichern(20);
47 $neuerKunde->zustandAusgeben();
48 $neuerKunde->speichern(40);
49 $neuerKunde->zustandAusgeben();
```

In Zeile 43 wird ein neuer Kunde erstellt. Es wird nur der Name des Kunden übergeben, nicht jedoch der verbrauchte Speicher. Damit wird dieser auf den Defaultwert 0 gesetzt, was auch der Aufruf der Methode `zustandAusgeben()` zeigt: Es ist 0 Speicher verbraucht. Dann folgen weitere Aufrufe zum Speichern und Freigeben von Speicher. Die Ausgabe zeigt Abbildung 9.2.

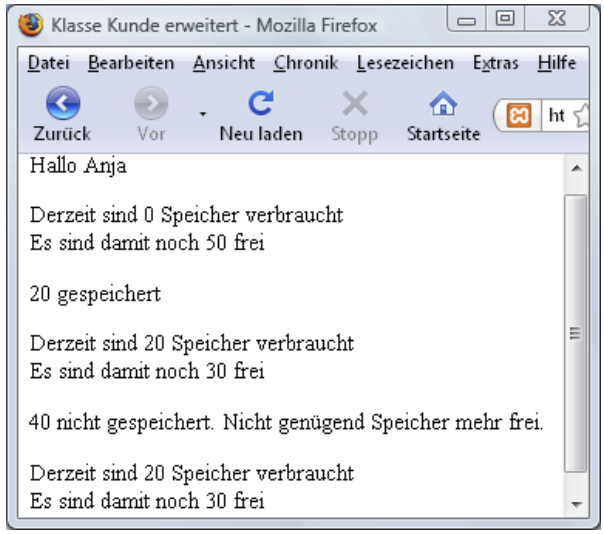


Abbildung 9.2: Die Ausgabe von Listing 9.6

9.6 Vererbung

Die Vererbung ist ein zentrales Konzept bei der objektorientierten Programmierung: Hierdurch können Sie Basisklassen erweitern und mit zusätzlichen Funktionen ausstatten.

9.6.1 Premiumkunden

Nehmen wir an, dass es neben den »normalen« Kunden auch noch Premiumkunden geben soll. Diesen steht mehr Speicherplatz zur Verfügung und außerdem haben sie mehr Konfigurationsmöglichkeiten und können etwa das Farbschema der Benutzeroberfläche bestimmen.

Dies realisieren Sie am besten über eine neue Klasse `Premiumkunde`. Diese Klasse hat sehr viele Gemeinsamkeiten mit der Klasse `Kunde`, aber weicht in ein paar Punkten ab. Es wäre jetzt unökonomisch, den Code der Klasse Kunde zu kopieren und mit den notwendigen Ergänzungen zu versehen. Denn wenn dann etwas an Eigenschaften oder Methoden der Klasse `Kunde` geändert werden, müssten Sie diese Änderungen an zwei Stellen durchführen – neben der Klasse `Kunde` auch an der Klasse `Premiumkunde`.

Genau für solche Fälle gibt es die Vererbung. Sie können eine neue Klasse erstellen, die die Eigenschaften und Methoden einer bestehenden Klasse erbt. Zusätzlich kann die neue Klasse aber dann weitere Eigenschaften definieren, neue Methoden implementieren oder bestehende überschreiben.

Um zu kennzeichnen, dass eine neue Klasse die Eigenschaften und Methoden von einer bestehenden Klasse erben soll, schreiben Sie hinter dem Namen der Klasse das Schlüsselwort `extends` gefolgt von der Klasse, von der sie erben soll.

```
class Premiumkunde extends Kunde
```

Damit Premiumkunden mehr Speicher erhalten, genügt jetzt folgende Klassendefinition:

```
class Premiumkunde extends Kunde
{
  public $speicherGesamt = 100;
}
```

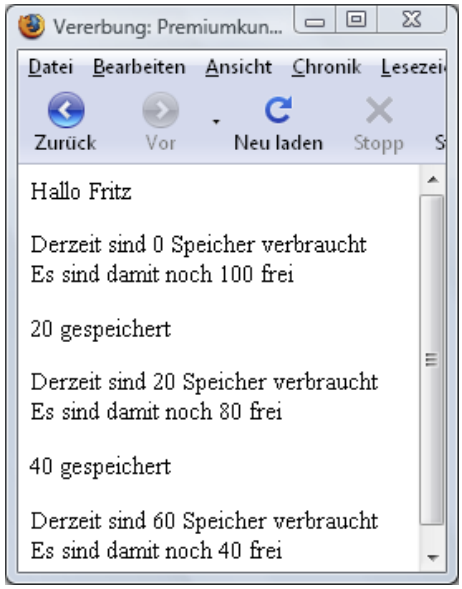


Abbildung 9.3: Ein Premiumkunde erbt alles vom normalen Kunden, hat aber mehr Speicherplatz.

Wenn Sie einen neuen Premiumkunden erstellen und wie eben die verschiedenen Methoden zum Speichern und Speicherfreigeben aufrufen, werden Sie sehen, dass es geklappt hat: Der Premiumkunde hat alle Eigenschaften und Methoden geerbt – allerdings hat er mehr Gesamtspeicherplatz.

Listing 9.7: Ein Premiumkunde (premiumkunde_beispiel.php)

```
01 require_once "kunde.php";
02 class Premiumkunde extends Kunde
03 {
04   public $speicherGesamt = 100;
05 }
06 $kunde2 = new Premiumkunde ("Fritz");
07 $kunde2->halloSagen();
08 $kunde2 ->zustandAusgeben();
09 $kunde2 ->speichern(20);
10 $kunde2 ->zustandAusgeben();
11 $kunde2 ->speichern(40);
12 $kunde2 ->zustandAusgeben();
```

Sie sehen, dass zu Beginn des Beispiels die Datei `kunde.php` eingebunden wird. Diese enthält die Klassendefinition der Klasse Kunde. Abbildung 9.3 zeigt die Ausgabe von Listing 9.7.

Im Beispiel gerade wurde die Eigenschaft `$speicherGesamt` überschrieben. Ebenso lassen sich auch Methoden überschreiben oder zusätzliche Eigenschaften und Methoden definieren.

Wenn der Premiumkunde das Aussehen seiner Benutzeroberfläche wählen kann und dies wird über den Konstruktor übergeben, können Sie auch den Konstruktor überschreiben:

```
01 class Premiumkunde extends Kunde
02 {
03   public $speicherGesamt= 100;
04   public $farbSchema;
05   public function __construct($name, $speicherVerbraucht = 0, $farbSchema =
       "Sonnenaufgang")
06   {
07     $this->name = $name;
08     $this->speicherVerbraucht = $speicherVerbraucht;
09     $this->farbSchema = $farbSchema;
10   }
11 }
```

Und bei der Erstellung eines neuen Kunden können dann drei Parameter übergeben werden.

Listing 9.8: Der Konstruktor wird in der abgeleiteten Klasse überschrieben (premiumkunde_beispiel_erw.php).

```
12 $kunde3 = new Premiumkunde ("Julian", 20, "Wüstenstimmung");
13 $kunde3->halloSagen();
14 echo "<br />Das gewählte Farbschema ist: {$kunde3->farbSchema}<br />";
15 $kunde3 ->zustandAusgeben();
```

Sie haben gesehen, wie die Klasse Premiumkunde die Eigenschaften und Methoden der Klasse Kunde erben und erweitern kann. Bei der ursprünglichen Klasse, also Kunde, spricht man auch von *Elternklasse*, bei der abgeleiteten von *Kindklasse*. Wie Sie gleich sehen werden, kann man innerhalb der abgeleiteten Klasse die Elternklasse über `parent` (engl. für Elternteil) ansprechen. Aber im Unterschied zu der Abstammung in der realen Welt, in der – zumindest meiner Erfahrung nach – Kinder immer die Eigenschaften zumindest von zwei Elternteilen erben, können in PHP Klassen immer nur von einer Elternklasse abgeleitet werden. Es gibt keine Mehrfachvererbung im Gegensatz beispielsweise zu C++. Sie können aber richtige Stammbäume erstellen, das heißt immer weiter und tiefer abgeleitete Klassen definieren.

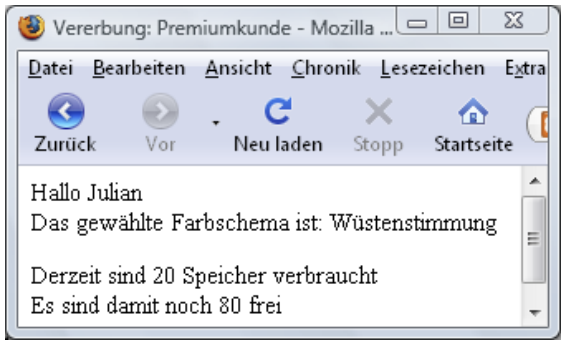


Abbildung 9.4: In der erweiterten Version können Premiumkunden auch ein Farbschema wählen.

Beim Entwurf von Klassen stellt sich mitunter die Frage, welche man als Elternklasse definieren soll und welche als abgeleitete. Eigentlich könnte man das Beispiel doch auch umgekehrt aufbauen, die Premiumkunden als ursprüngliche Klasse implementieren und die Kundenklasse davon ableiten?

Nehmen Sie das `extends` zur Ableitung einer Klasse ruhig einmal wörtlich: Es heißt »erweitert«. Die *abgeleitete Klasse erweitert die Basisklasse* und bietet mehr und speziellere Funktionalität – und dann ist klar, dass die Basisklasse der Kunde ist und der Premiumkunde abgeleitet und nicht umgekehrt. Prinzipiell geht es bei der Vererbung immer vom Allgemeineren zum Spezielleren.

Hinweis

Noch ein weiteres Beispiel für Basisklassen und abgeleitete Klassen: Wenn Sie PHP-Frameworks zur Arbeitserleichterung einsetzen, so werden Sie sehen, dass üblicherweise die Frameworks Ihnen die Basisklassen automatisch generieren, Ihre Änderungen notieren Sie hingegen in den abgeleiteten Klassen.

9.6.2 Konstruktoren in der Basisklasse und in der abgeleiteten Klasse

Noch einmal zum Konstruktor in Eltern- und abgeleiteten Klassen. In Listing 9.7 haben Sie gesehen, dass in abgeleiteten Klassen prinzipiell der Konstruktor der Elternklasse aufgerufen wird. Der Konstruktor der Elternklasse wird jedoch nicht aufgerufen, wenn Sie ihn in der abgeleiteten Klasse überschrieben haben. Wenn Sie dann trotzdem den Konstruktor der Elternklasse aufrufen möchten, so verwenden Sie `parent::__construct()`, also das Schlüsselwort `parent` in Kombination mit Gültigkeitsbereichsoperator `::` und `__construct()`. Das folgende Beispiel demonstriert noch einmal, wie das mit dem Konstruktor in Eltern- und Kindklasse ist:

Listing 9.9: Auch der Konstruktor der Elternklasse kann in der abgeleiteten Klasse aufgerufen werden (aufruf_konstr_elternklasse.php).

```
01 class Elternklasse
02 {
03   public function __construct()
04   {
05     echo "In " . get_class($this);
06     echo ": Aufruf des Konstruktors der Elternklasse<br />\n";
07   }
08 }
09
10 class Kindklasse1 extends Elternklasse
11 {
12
13 }
14
15 class Kindklasse2 extends Elternklasse
16 {
17   public function __construct()
18   {
19     echo "In " . get_class($this);
20     echo ": Aufruf des Konstruktors der Kindklasse<br />\n";
21     parent::__construct();
22   }
23 }
24
25 $objekt1 = new Elternklasse();
26 $objekt2 = new Kindklasse1();
27 $objekt3 = new Kindklasse2();
```

Im Beispiel wird in Zeile 1 eine Klasse mit Namen `Elternklasse` definiert. Sie enthält einen Konstruktor, in dem die aktuelle Klasse und der Text »Aufruf des Konstruktors der Elternklasse« ausgegeben werden.

Die Klasse `Kindklasse1` in Zeile 10 erweitert die `Elternklasse`, definiert aber nichts Neues. Eine zweite Klasse `Kindklasse2` in Zeile 15 enthält hingegen einen eigenen Konstruktor, der den Namen der aktuellen Klasse und eine Meldung ausgibt. Außerdem wird im Konstruktor von `Kindklasse2` der Konstruktor der Elternklasse aufgerufen (Zeile 21). Schließlich werden in Zeile 25–27 Objekte von allen Klassen erstellt.

Abbildung 9.5 zeigt, dass in `Kindklasse1` automatisch der Konstruktor der Elternklasse aufgerufen wird. In `Kindklasse2` muss dieser Aufruf hingegen explizit geschehen und dann werden beide Konstruktoren – der Elternklasse und der Kindklasse ausgeführt. Wenn Sie einmal testweise Zeile 21 im obigen Listing auskommentieren, sehen Sie, dass der Aufruf des Konstruktors der Elternklasse nicht mehr automatisch geschieht, wenn der Konstruktor in der abgeleiteten Klasse überschrieben ist.

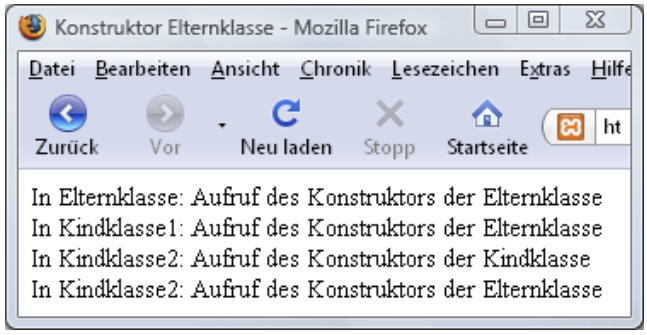


Abbildung 9.5: Ist der Konstruktor der Elternklasse in der abgeleiteten Klasse überschrieben, kann er über parent::__construct() explizit aufgerufen werden.

Tipp

Im Beispiel haben Sie gesehen, wie Sie über `parent::__construct()` die Konstruktormethode der Elternklasse aufrufen. `parent` können Sie aber auch zum Aufruf beliebiger anderer Methoden der Elternklasse nutzen.

9.7 Zugriff steuern

Derzeit haben wir bei der Klasse `Kunde` eigene Methoden erstellt, um den verbrauchten Speicher zu ermitteln. Ungeachtet dessen kann man aber den verbrauchten Speicher auch direkt setzen – oder sogar den Gesamtspeicher.

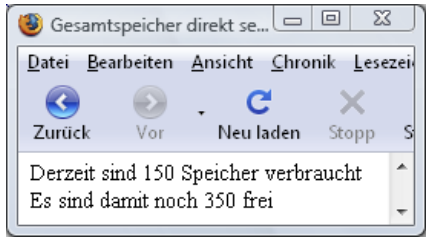


Abbildung 9.6: Noch 350 frei – das ist eigentlich nicht vorgesehen.

Listing 9.10: Eigenschaften wie der Gesamtspeicher können direkt gesetzt werden (kunde_speicher_direkt.php).

```
require_once "kunde.php";
$neuerKunde = new Kunde("Anja");
$neuerKunde->speicherGesamt = 500;
$neuerKunde->speicherVerbraucht = 150;
$neuerKunde->zustandAusgeben();
```

Das stellt das ganze System infrage und widerspricht dem Grundgedanken, wie die Klasse entworfen wurde. Um dem entgegenzuwirken, müssen Sie den *direkten Zugriff auf bestimmte Methoden/Eigenschaften einschränken*. Genau hierfür gibt es die Schlüsselwörter `private`, `protected` und `public`.

Bisher stand bei Methoden und Eigenschaften immer `public`. `public` erlaubt den Zugriff aus dem Objektkontext und auch aus abgeleiteten Klassen heraus, also kurz gesagt immer. Bei Eigenschaften und Methoden gibt es Unterschiede bei den Zugriffsmodifizierern.

- Wenn Sie bei einer Methode keinen Zugriffsmodifizierer angeben, wird diese als `public` behandelt.
- Bei Eigenschaften müssen Sie etwas angeben. Ebenfalls erlaubt ist das noch in PHP-Version 4 gebräuchliche `var` anstatt von `public`. Das bedeutet dann ebenfalls: freien Zugriff von überall.

`private` und `protected` schränken hingegen den Zugriff ein. Auf eine als `private` oder `protected` deklarierte Eigenschaft/Methode können Sie nicht von einem Objekt her zugreifen.

Durch Listing 9.11 erhalten Sie die in Abbildung 9.7 gezeigte Fehlermeldung.

Listing 9.11: Das ist nicht erlaubt: Zugriff auf eine private Eigenschaft (zugriffsfehler.php).

```
class Beispielklasse
{
  private $eigenschaft = 50;
}
$objekt = new Beispielklasse();
$objekt->eigenschaft = 500;
```

Abbildung 9.7: Fehlermeldung bei direktem Zugriff auf eine als private gekennzeichnete Eigenschaft

Genauso wird ein Fehler gemeldet, wenn Sie die Eigenschaft als `protected` deklarieren und versuchen, über das Objekt direkt zuzugreifen. Der folgende Code erzeugt ebenfalls eine Fehlermeldung:

```
class Beispielklasse
{
  protected $eigenschaft = 50;
}
$objekt = new Beispielklasse();
$objekt->eigenschaft = 500;
```

Im Objektkontext gibt es keinen Unterschied zwischen `private` und `protected`. Bei abgeleiteten Klassen hingegen ist ein Unterschied festzustellen: Eine als `protected` gekennzeichnete Methode/Eigenschaft kann in der abgeleiteten Klasse abgerufen werden, eine als `private` gekennzeichnete Methode/Eigenschaft ist in der abgeleiteten Klasse *hingegen nicht sichtbar*. Das ist ein wichtiger Punkt: Das bedeutet nämlich, dass man in der abgeleiteten Klasse eine Methode/Eigenschaft mit demselben Namen definieren kann, aber nicht auf die ursprüngliche zugreifen kann.

Ein Beispiel hierzu: In der `BeispielklasseA` wird `$eigenschaft` als `private` gekennzeichnet.

```
01 class BeispielklasseA
02 {
03   private $eigenschaft = 50;
04 }
```

Die `BeispielklasseB` ist von `BeispielklasseA` abgeleitet. In ihr gibt es eine Methode mit Namen `test()`, in der versucht wird, auf die als `private` gekennzeichnete Eigenschaft der Elternklasse zuzugreifen.

Listing 9.12: Versuch, in einer abgeleiteten Klasse auf die als private gekennzeichnete Eigenschaft der Elternklasse zuzugreifen (protected_private.php)

```
05 class BeispielklasseB extends BeispielklasseA
06 {
07   public function test() {
08     echo "Versuch Zugriff auf private-Eigenschaft ";
```

```
09     echo $this->eigenschaft;
10   }
11 }
12 $objekt = new BeispielklasseB();
13 $objekt->test();
```

Wie zu erwarten, funktioniert der Zugriff nicht, denn als `private` gekennzeichnete Eigenschaften sind in abgeleiteten Klassen nicht sichtbar. Jedoch erzeugt der Zugriff auf die unbekannte Eigenschaft der Elternklasse *keinen fatalen Fehler* – sondern genau wie sonst auch der Einsatz einer nicht-initialisierten Variablen einen Hinweis (Notice), sofern Sie den Fehlermeldungslevel entsprechend eingestellt haben.

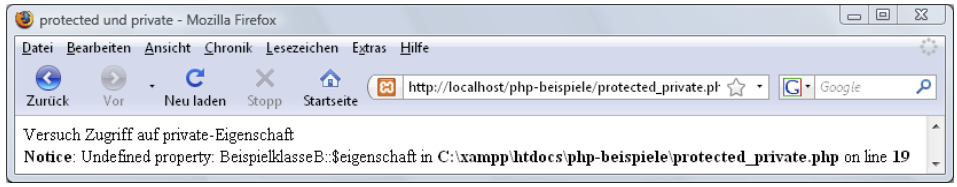


Abbildung 9.8: Der Zugriff auf eine private Eigenschaft der Elternklasse erzeugt nur eine Notice.

Das Ganze klappt hingegen, wenn die Eigenschaft in der Elternklasse als `protected` gekennzeichnet ist.

Listing 9.13: So geht's: In der abgeleiteten Klasse kann man auf die protected Eigenschaft der Elternklasse zugreifen (protected_private_mod.php).

```
01 class BeispielklasseA
02 {
03   protected $eigenschaft = 50;
04 }
05 class BeispielklasseB extends BeispielklasseA
06 {
07   public function test() {
08     echo "Versuch Zugriff auf protected-Eigenschaft ";
09     echo $this->eigenschaft;
10   }
11 }
12 $objekt = new BeispielklasseB();
13 $objekt->test();
```

Ebenfalls möglich ist es allerdings, eine `private` Eigenschaft der Elternklasse in der abgeleiteten Klasse neu zu definieren (Listing 9.14).

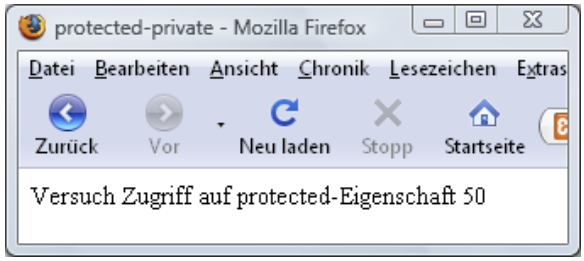


Abbildung 9.9: Der Zugriff auf eine als protected deklarierte Eigenschaft klappt in der abgeleiteten Klasse.

Listing 9.14: Das hingegen geht: Eine private Eigenschaft der Elternklasse kann in der abgeleiteten Klasse neu definiert werden (private_ueberschreiben.php).

```
01 class BeispielklasseA
02 {
03   private $eigenschaft = 50;
04 }
05
06 class BeispielklasseB extends BeispielklasseA
07 {
08   protected $eigenschaft = 20;
09   public function test() {
10     echo " Zugriff auf Eigenschaft ";
11     echo $this->eigenschaft;
12   }
13 }
14 $objekt = new BeispielklasseB();
15 $objekt->test();
```

Wie zu erwarten, wird dann hier 20 ausgegeben – der in der abgeleiteten Klasse gesetzte Wert.

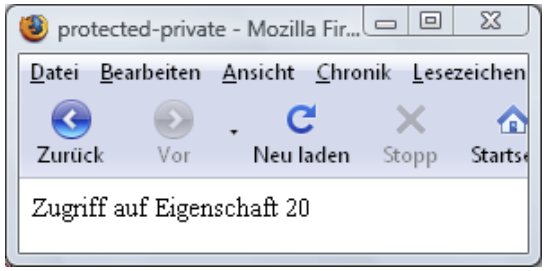


Abbildung 9.10: Der als private deklarierte Wert in der Elternklasse kann in der Kindklasse überschrieben werden.

Kehren wir noch einmal zurück zum Beispiel mit der Klasse `Kunde` und `Premiumkunde`. Hier ist es auf jeden Fall sinnvoll, den direkten Zugriff auf `$speicherGesamt` und `$speicherVerbraucht` über `protected` zu verhindern. Alle anderen Eigenschaften/Methoden müssen jedoch `public` bleiben, da ein direkter Zugriff vorgesehen ist.

Listing 9.15: Hier sehen Sie nur die Namen von Eigenschaften und Methoden mit den Zugriffsmodifizierern. Das vollständige Beispiel finden Sie unter kunden_mit_zugriffsmod.php.

```
class Kunde
{
  public $name;
  protected $speicherGesamt;
  protected $speicherVerbraucht;

  public function __construct()
  public function halloSagen()

  public function speichern()
  public function speicherFreigeben()
  public function zustandAusgeben()
 }
class Premiumkunde extends Kunde
{
  protected $speicherGesamt;
  public $farbSchema;
  public function __construct()
}
```

Dieses Beispiel ließe sich jetzt noch weiter verfeinern, indem der direkte Zugriff auf weitere Eigenschaften wie `$name` oder `$farbSchema` ebenfalls über den Zugriffsmodifizierer `protected` unterbunden wird. Dann müsste man zusätzliche Methoden implementieren, über die eine Ausgabe möglich ist. Ein Beispiel, bei dem das auch so gehandhabt wird, ist die in PHP vordefinierte Klasse `Exception` (siehe Abschnitt 9.14).

Hinweis

Durch diese Zugriffsmodifizierer ist eine Kapselung von Informationen möglich. Wenn Sie eine Klasse benutzen, müssen Sie nur wissen, welche öffentlichen Methoden und Eigenschaften es gibt und wozu diese da sind. Wie sie implementiert sind und was es darüber hinaus noch an nicht-öffentlichen Methoden und Eigenschaften gibt, ist für Sie nicht relevant. Außer Sie wollen die Klasse erweitern.

9.8 Vererbung und Überschreibung genau steuern

Gerade haben Sie gesehen, wie Sie den Zugriff auf Methoden und Eigenschaften steuern können. Jetzt geht es darum, wie sich steuern lässt, ob und wie welche Eigenschaften/Methoden überschrieben werden können.

9.8.1 Überschreibung verhindern mit final

Wichtige Methoden können durch `final` geschützt werden.

final bei Methoden

Wenn Sie in einer Klasse Methoden implementiert haben, die für den allgemeinen Ablauf sehr wichtig sind, können Sie mit `final` verhindern, dass diese in abgeleiteten Klassen überschrieben werden. So könnten Sie beispielsweise in der Kunde-Klasse die Methoden `speichern()` und `speicherFreigeben()` als `final` deklarieren. Dann können sie in der abgeleiteten Klasse nicht überschrieben und damit auch nicht verändert werden. `final` wird ganz am Anfang, d. h. vor dem Zugriffsmodifizierer notiert.

```
01 class Kunde
02 {
03 /* alles andere wie gehabt */
04   final public function speichern($speicherBedarf)
05   {
06     if (($this->speicherGesamt - $this->speicherVerbraucht)
07         >= $speicherBedarf) {
08        $this->speicherVerbraucht=
09         $this->speicherVerbraucht+$speicherBedarf;
10        echo "$speicherBedarf gespeichert";
11      } else {
12        echo "Nicht gespeichert. Nicht genügend Speicher mehr frei.";
13      }
14   }
15   final public function speicherFreigeben($speicher)
16   {
17     $this->speicherVerbraucht= $this->speicherVerbraucht -
18     $speicher;
19     echo "$speicher Speicher frei gegeben";
20   }
21 /* alles andere wie gehabt */
22 }
```

Versuchen Sie jetzt, die Methoden zu überschreiben, erhalten Sie eine Fehlermeldung.

Listing 9.16: Als final deklarierte Methoden können in abgeleiteten Klassen nicht überschrieben werden (final.php).

```
23 class PremiumKunde extends Kunde
24 {
25 /* alles andere wie gehabt */
26   public function speicherFreigeben($speicher) {
27     echo "doch überschrieben";
28   }
29 }
30 $kunde3 = new PremiumKunde ("Julian", 20, "Wüstenstimmung");
31 $kunde3->halloSagen();
32 echo "<br />Das gewählte Farbschema ist: {$kunde3->farbSchema}<br />";
33 $kunde3 ->zustandAusgeben();
34 $kunde3 ->speichern(20);
```

Abbildung 9.11: Fehlermeldung beim Versuch, eine als final deklarierte Methode zu überschreiben.

`final` ist also wichtig, um sicherzustellen, dass etwas 1:1 in die abgeleitete Klasse übernommen wird. Umgekehrt sollten Sie daran denken, wenn Sie eine fremde Klasse verwenden, dass Sie keinerlei Änderungen in Ihrer abgeleiteten Klasse an den mit `final`-deklarierten Methoden vornehmen können.

final bei Klassen

Eben ging es darum, welche Wirkung `final` bei einer Methode hat. Wenn Sie hingegen eine Klasse als `final` deklarieren, sorgen Sie dafür, dass sie nicht abgeleitet werden kann. Das folgende Listing gibt deswegen eine Fehlermeldung aus.

Listing 9.17: Eine Klasse wird als final gekennzeichnet (finale_klasse.php).

```
final class Endgueltig
{
}
class Versuch extends Endgueltig
{
}
$obj = new Versuch();
```

Wie Sie gesehen haben, können Sie `final` bei Methoden und Klassen einsetzen, bei Eigenschaften wäre es hingegen nicht sinnvoll.

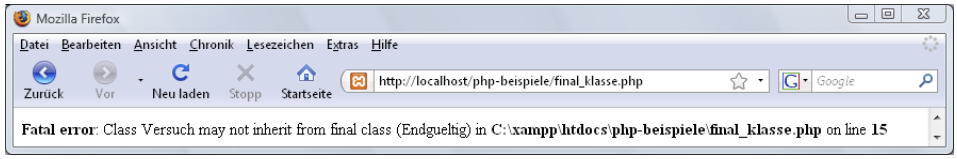


Abbildung 9.12: Beim Versuch, eine als final gekennzeichnete Klasse abzuleiten, wird eine Fehlermeldung ausgegeben.

9.8.2 Überschreibung fordern mit abstract

Auch das Schlüsselwort `abstract` ist wichtig bei der Vererbung. Es macht aber genau das Gegenteil von `final`: Eine Methode, die als `abstract` deklariert wird, *muss* in einer abgeleiteten Klasse *überschrieben werden*.

Zwei Punkte gibt es dazu zu beachten:

- Wenn eine Klasse auch nur eine als `abstract` deklarierte Methode erhält, können Sie von dieser Klasse nicht direkt ein Objekt erzeugen, sondern müssen erst eine Klasse ableiten, in der Sie die Methode implementieren.
- Außerdem muss die gesamte Klasse als `abstract` definiert werden, wenn sie auch nur eine abstrakte Methode enthält.

Ein Beispiel: Die Klasse `GanzAbstract` wird als `abstract` deklariert, da sie eine abstrakte Methode `begruessen()` beinhaltet.

```
01 abstract class GanzAbstrakt
02 {
03   abstract public function begruessen();
04   public function ausgabe()
05   {
06     echo "nicht abstrakt";
07   }
08 }
```

Abstrakte Methoden erhalten das Schlüsselwort `abstract` und nur den Namen – ganz ohne Funktionsrumpf, selbst leere geschweifte Klammern `{}` würden eine Fehlermeldung erzeugen.

Dann wird eine neue Klasse erstellt, die von der Klasse `GanzAbstrakt` abgeleitet wird. Sie implementiert die Methode `begruessen()`.

```
09 class NeueKlasse extends GanzAbstrakt
10 {
11   public function begruessen()
12   {
13     echo "Schönen Tag auch!";
14   }
15 }
```

Von der abgeleiteten Klasse kann dann ein neues Objekt erstellt werden. Direkt ein Objekt auf der Basis der abstrakten Klasse zu erstellen, ist hingegen nicht möglich.

Listing 9.18: Von einer abstrakten Klasse kann nicht direkt ein Objekt erzeugt werden (abstract.php).

```
16 /* das geht nicht: */
17 /*$beispiel = new GanzAbstrakt;
18 $beispiel->begruessen(); */
19
20 /* das geht */
21 $beispiel = new NeueKlasse();
22 $beispiel->begruessen();
```

Abbildung 9.13: Wenn Sie versuchen, direkt eine Instanz einer abstrakten Klasse zu erstellen, erhalten Sie eine Fehlermeldung.

9.8.3 Schnittstellen – Interfaces

Gerade haben Sie gesehen, dass man von abstrakten Klassen nicht direkt Objekte erstellen kann. Das gilt ebenso für die nun vorgestellten Schnittstellen. Diese gehen jedoch noch einen Schritt weiter als abstrakte Klassen: In Schnittstellen definieren Sie, welche Methoden zu implementieren sind, ohne jedoch vorzugeben wie. Nützlich sind Schnittstellen, um die Funktionalität von mehreren Klassen zusammenzufassen.

Eine Schnittstelle erstellen Sie mit dem Schlüsselwort `interface` anstelle von `class`. Alle Methoden, Eigenschaften und Konstanten von Schnittstellen müssen öffentlich, d. h. `public` sein. Um die Schnittstelle anzuwenden, verwenden Sie das Schlüsselwort `implements` anstelle von `extends`.

Im folgenden Listing sehen Sie die Definition eines Interfaces `Schreiben` mit drei öffentlichen Methoden ohne Methodenrumpf. Die Klasse `Dateischreiben` implementiert dieses Interface, d. h. sie definiert die einzelnen Methoden. Von dieser Klasse kann dann ganz normal ein Objekt erstellt und die Methoden aufgerufen werden.

Listing 9.19: Die Definition und Implementierung eines Interfaces (interface.php)

```
01 interface Schreiben
02 {
03   public function oeffnen();
04   public function schreiben();
```

```
05   public function schliessen();
06 }
07 class Dateischreiben implements Schreiben
08 {
09    public function oeffnen()
10   {
11     echo "Datei geöffnet. ";
12   }
13   public function schreiben()
14   {
15     echo "In Datei schreiben. ";
16   }
17   public function schliessen()
18   {
19     echo "Datei geschlossen. ";
20   }
21 }
22 $beispiel = new Dateischreiben();
23 $beispiel->oeffnen();
24 $beispiel->schreiben();
25 $beispiel->schliessen();
```

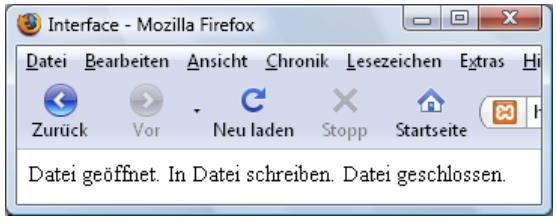


Abbildung 9.14: Die Ausgabe von Listing 9.19

Wichtig ist dabei, dass in der Klasse, die das Interface implementiert, wirklich *alle* Methoden definiert werden. Fehlt eine, erhalten Sie eine Fehlermeldung.

Tipp

Eine Klasse kann gleichzeitig von einer anderen Klasse erben und ein bestimmtes Interface implementieren:

```
class A extends B implements C
```

Außerdem kann eine Klasse auch mehrere Interfaces implementieren, vorausgesetzt, dass sie nicht gleich lautende Methodennamen besitzen:

```
class D implements E, F
```

Und Interfaces können auch vererbt werden.

9.9 Type Hints

Angenommen, `oeffnen()` und `schreiben()` der Klasse `Dateischreiben` sind zwei Methoden, die Sie häufig zusammen brauchen. Dann können Sie sich eine Funktion schreiben, die dafür sorgt, dass beide in der richtigen Reihenfolge ausgeführt werden:

```
function direktschreiben($obj)
{
  $obj->oeffnen();
  $obj->schreiben();
}
```

`direktschreiben()` funktioniert allerdings nur, wenn Sie ihr Objekte übergeben, die die entsprechenden Methoden auch implementieren: Das heißt, die Objekte müssen von der Klasse `Dateischreiben` stammen oder von einer anderen Klasse, die das Interface `Schreiben` implementiert. Dies können Sie über *Type Hints* angeben. Dazu schreiben Sie vor den bei der Funktion übergebenen Parameter den Namen der Klasse, von dem das Objekt eine Instanz sein soll, oder geben das Interface an.

```
function direktschreiben(Schreiben $obj)
```

Im Zusammenhang sieht das dann so aus:

Listing 9.20: Über Type Hints lässt sich sicherstellen, dass nur Objekte einer bestimmten Klasse übergeben werden (type_hint.php).

```
01 interface Schreiben
02 {
03   /* wie oben */}
04 class Dateischreiben implements Schreiben
05 {
06 /* wie oben */
07 }
08
09 $beispiel = new Dateischreiben();
10 function direktschreiben(Schreiben $obj)
11 {
12   $obj->oeffnen();
13   $obj->schreiben();
14 }
15 direktschreiben($beispiel);
```

Wenn Sie hingegen die Funktion `direktschreiben()` mit einem nicht passenden Objekt aufrufen, erhalten Sie eine Fehlermeldung.

Listing 9.21: Mit einer Instanz einer anderen Klasse funktioniert es nicht (type_hint_fehler.php).

```
01 class Anders
02 {
03 }
04 $test = new Anders();
05 function direktschreiben(Schreiben $obj)
06 {
07   $obj->oeffnen();
08   $obj->schreiben();
09 }
10 direktschreiben($test);
```

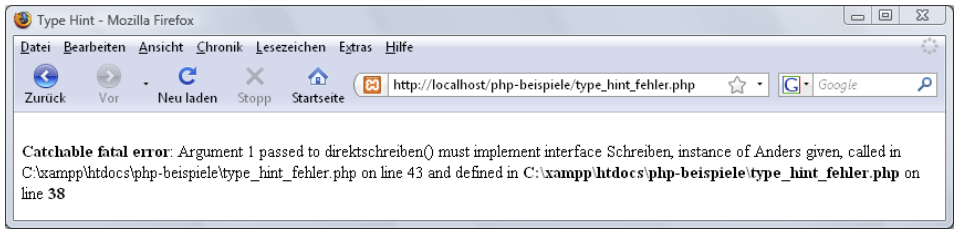


Abbildung 9.15: Wird nicht das richtige Objekt übergeben, erhalten Sie eine Fehlermeldung.

9.10 static – auch ohne Objekt aufrufbar

Das nächste Schlüsselwort, was vorgestellt werden soll, ist `static`.

9.10.1 Statische Methoden

Statische Methoden können Sie aufrufen, ohne ein Objekt erstellt zu haben. Erinnern Sie sich an das Beispiel zur Ausgabe eines Datums mit deutschen Wochentagen und Monatsnamen aus Kapitel 6? Hier sehen Sie es noch einmal:

Listing 9.22: Datum ausgeben lassen (date_deutsch_00.php)

```
01 class Datum
02 {
03   public function ausgeben()
04   {
05     date_default_timezone_set("Europe/Berlin");
06     $tage = array (
07         "Mon" => "Montag",
08         "Tue" => "Dienstag",
09         "Wed" => "Mittwoch",
```

```
10          "Thu" => "Donnerstag",
11          "Fri" => "Freitag",
12          "Sat" => "Samstag",
13          "Sun" => "Sonntag");
14      $wochentag = $tage[date("D")];
15      $monate = array (
16          "Jan" => "Januar",
17          "Feb" => "Februar",
18          "Mar" => "März",
19          "Apr" => "April",
20          "Mai" => "Mai",
21          "Jun" => "Juni",
22          "Jul" => "Juli",
23          "Aug" => "August",
24          "Sep" => "September",
25          "Oct" => "Oktober",
26          "Nov" => "November",
27          "Dec" => "Dezember");
28     $monat = $monate[date("M")];
29     $wochentag = $tage[date("D")];
30     echo "$wochentag, den ";
31     echo date("j. " ) .$monat . date(" Y");
32    }
33 }
34 $heute = new Datum();
35 $heute->ausgeben();
```

Um das aktuelle Datum auszugeben, müssen Sie wie gewohnt ein neues Objekt der Klasse initialisieren und dann die gewünschte Methode aufrufen:

```
$heute = new Datum();
$heute->ausgeben();
```

Einfacher geht es durch `static`. Wenn Sie die Methode `ausgeben()` als `static` deklarieren, können Sie auf sie zugreifen, ohne ein Objekt der Klasse erstellen zu müssen.

Ergänzen Sie zuerst bei der Methode

```
public static function ausgeben()
```

Dann können Sie diese Methode direkt aufrufen:

Listing 9.23: Direkter Zugriff auf statische Methoden (static.php)

```
Datum::ausgeben();
```

9.10.2 Statische Eigenschaften

Neben Methoden können auch Eigenschaften als statisch deklariert werden.

Listing 9.24: Auch Eigenschaften können als statisch deklariert werden (static_eigenschaft.php).

```
01 class Beispiel
02 {
03   public static $zahl = 42;
04   public function ausgeben()
05   {
06     return self::$zahl;
07   }
08 }
09 echo Beispiel::$zahl;
10 /*Folgende Zeile erzeugt einen Hinweis, wenn Fehlerlevel Strict */
11 /*echo Beispiel::ausgeben();*/
12 echo "<br />\n";
13
14 $bsp = new Beispiel();
15 echo $bsp->ausgeben();
```

Im Beispiel sehen Sie, dass die Eigenschaft `$zahl` in der Klasse Beispiel als statisch deklariert wird. Dann können Sie direkt darauf zugreifen über `Beispiel::$zahl`. Außerdem gibt es die Methode `ausgeben()`, die zeigt, wie Sie innerhalb der Klasse über `self` auf die statische Eigenschaft zugreifen können.

Wenn Sie versuchen, auf die nicht statisch definierte Methode direkt zuzugreifen, erhalten Sie bei entsprechendem Fehlermeldungslevel einen Hinweis. Sie können aber ein neues Objekt initiieren und dann wie gewohnt auf die Methode zugreifen.

Statische Eigenschaften ähneln den Klassenkonstanten (siehe Abschnitt 9.4). Aber im Unterschied zu den Klassenkonstanten, auf die Sie nur lesend zugreifen können, können Sie statische Eigenschaften auch verändern:

```
Beispiel::$zahl = 45;
```

Tipp

Statische Eigenschaften können dazu verwendet werden, um zu zählen, wie viele Instanzen einer Klasse erstellt wurden:

Listing 9.25: Instanzen lassen sich über statische Eigenschaften zählen (instanzenzaehler.php).

```
01 class Beispiel
02 {
03   static $count = 0;
```

```
04   public function __construct()
05   {
06     self::$count++;
07     echo "Nummer der Instanz: " . self::$count;
08   }
09 }
10 $a = new Beispiel();
11 echo "<br />\n";
12 $b = new Beispiel();
```

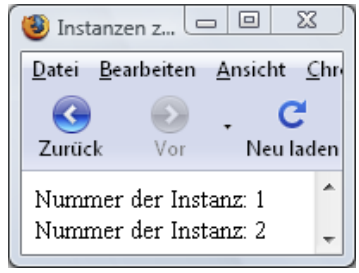


Abbildung 9.16: Die Instanzen werden gezählt.

9.10.3 Late Static Binding

Late Static Binding ist ein neu in PHP 5.3 eingeführtes Feature. Es betrifft die Behandlung von statischen Methoden oder Eigenschaften bei der Vererbung. Sehen wir uns ein Beispiel an:

Listing 9.26: Aufruf der überschriebenen statischen Kindklasse klappt nicht (ohne_late_static_binding.php).

```
01 class Elternklasse {
02   public function test() {
03     self::ausgabe();
04   }
05
06   public static function ausgabe() {
07     echo "Elternklasse: Klassenname ist: " . __CLASS__;
08   }
09 }
10
11 class Kindklasse extends Elternklasse {
12   public static function ausgabe() {
13     echo "Abgeleitete Klasse: Klassenname ist: " . __CLASS__;
14   }
15 }
16
```

```
17 $a = new Elternklasse();
18 $a->test();
19 echo "<br />\n";
20 $b = new Kindklasse();
21 $b->test();
```

Hier wird eine Elternklasse mit zwei Methoden definiert: Die Methode `test()` in Zeile 2 ruft die statische Methode `ausgabe()` auf. Die statische Methode `ausgabe()` (Zeile 6) gibt einen Text aus und die aktuelle Klasse über die vordefinierte Konstante `__CLASS__`.

In Zeile 11 wird eine Kindklasse definiert, die von der Elternklasse abgeleitet ist. Sie überschreibt die statische Methode `ausgabe()` der Elternklasse – ein etwas anders lautender Text und Klassennamen werden ausgegeben.

In Zeile 17 wird eine neue Instanz der Elternklasse erstellt und die Methode `test()` aufgerufen. Dann wird in Zeile 20 eine neue Instanz der Kindklasse definiert und hier ebenfalls die Methode `test()` aufgerufen. Wie Sie aber in der Ausgabe in Abbildung sehen, wird in beiden Fällen die statische Methode `ausgabe()` der Elternklasse aufgerufen.

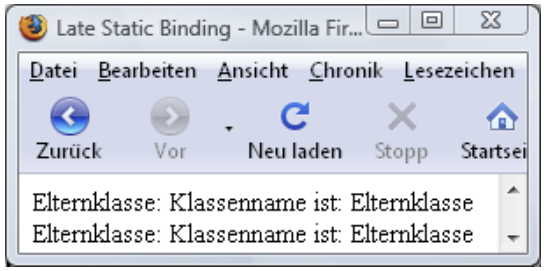


Abbildung 9.17: Beide Male wird die Methode ausgabe() der Elternklasse aufgerufen.

Das liegt daran, dass sich `self` auf die aktuelle Klasse bezieht, in der es steht, nicht auf die Klasse, von der aus die Methode aufgerufen wurde. `self` wird nämlich beim Kompilieren durch den Namen der Klasse ersetzt, in der es steht. Und damit erfolgt auch, wenn die Methode `test()` von der Kindklasse aufgerufen wurde, ein Aufruf von `Elternklasse::ausgabe()`.

Was hier fehlt, ist eine Möglichkeit, auf die Klasse zuzugreifen, die die Methode aufgerufen hat. Dies ist durch den Einsatz von `static` möglich. Dieses wird später, d. h. nicht zur Kompilierzeit, sondern zur Laufzeit, durch den Namen der aufrufenden Klasse ersetzt.

Listing 9.27: So klappt der Zugriff auf die statische Methode der Kindklasse (late_static_binding.php).

```
01 class Elternklasse {
02   public function test() {
03     static::ausgabe();
```

```
04   }
05
06   public static function ausgabe() {
07     echo "Elternklasse: Klassenname ist: " . __CLASS__;
08   }
09 }
10
11 class Kindklasse extends Elternklasse {
12   public static function ausgabe() {
13     echo "Abgeleitete Klasse: Klassenname ist: " . __CLASS__;
14   }
15 }
16
17 $a = new Elternklasse();
18 $a->test();
19 echo "<br />\n";
20 $b = new Kindklasse();
21 $b->test();
```

Jetzt sieht die Ausgabe aus wie gewünscht – beim Aufruf von `$b->test()` wird die statische Methode der Kindklasse aufgerufen.

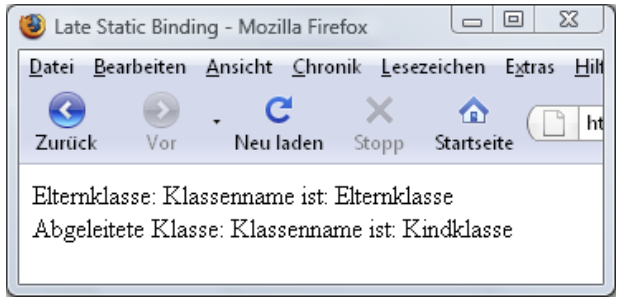


Abbildung 9.18: Wie gewünscht wird die statische Methode der Kindklasse aufgerufen.

9.11 Mehr magische Methoden

Formal erkennen Sie magische Methoden an den beiden Unterstrichen (__) am Anfang. Zwei magische Methoden haben Sie bereits kennen gelernt – nämlich die Konstruktor- und die Dekonstruktor-Methode (siehe Abschnitt 9.2), die automatisch aufgerufen werden, wenn ein neues Objekt erstellt bzw. ein Objekt zerstört wird. Nun zu weiteren magischen Methoden.

9.11.1 __set() und __get()

Die beiden magischen Methoden `__set()` und `__get()` werden automatisch aufgerufen, wenn ein Zugriff auf eine nicht definierte Eigenschaft erfolgt: `__set()` bei einem

schreibenden, `__get()` bei einem lesenden Zugriff. Wenn Sie `__set()` und `__get()` in Ihrer Klasse implementieren, können Sie festlegen, was in diesem Fall geschehen soll.

Listing 9.28: Über __get() und __set() definiert man, was bei einem Zugriff auf eine nicht vorhandene Eigenschaft geschehen soll (get_set.php).

```
01 class Beispiel
02 {
03   public $vorname;
04   public function halloSagen()
05   {
06     echo "Hallo {$this->vorname}";
07   }
08   public function __set($eigenschaft, $wert)
09   {
10     if ($eigenschaft == "alter") {
11       echo "Das Alter ist hier nicht relevant";
12     } else {
13       echo "Sie wollen die Eigenschaft $eigenschaft setzen, die es nicht gibt.";
14     }
15   }
16   public function __get($eigenschaft)
17   {
18     echo "Die Eigenschaft $eigenschaft gibt es nicht.";
19   }
20 }
21
22 $bsp = new Beispiel();
23 $bsp->vorname = "Amina";
24
25 $bsp->halloSagen();
26 echo "<br />\n";
27 $bsp->alter = 2;
28 echo "<br />\n";
29 echo $bsp->name;
```

In der Klasse `Beispiel` wird die Eigenschaft `$vorname` (Zeile 3) und die Methode `halloSagen()` implementiert. In Zeile 8 steht die magische Methode `__set()`, die bestimmt, was passieren soll, wenn eine Eigenschaft, die nicht definiert ist, gesetzt wird. `__set()` erwartet zwei Parameter – zuerst die Eigenschaft, die gesetzt wird und dann den Wert, auf den die Eigenschaft gesetzt wird. Beide können Sie innerhalb der Methode auslesen. Im Beispiel wird gezeigt, dass man je nach gesetzter Eigenschaft auch unterschiedliche Dinge tun kann: Dafür überprüfen Sie, welchen Wert die Eigenschaft hat. Im Beispiel wird geprüft, ob es sich dabei um die Eigenschaft `$alter` handelt und in diesem Fall eine entsprechende Meldung ausgegeben.

Tipp

Wenn Sie hier mehr Fälle prüfen wollten, sollten Sie die `switch`-Konstruktion benutzen.

In Zeile 16 wird die magische `__get()`-Methode implementiert: Sie erwartet nur einen Parameter: die Eigenschaft, die ausgelesen wird. Auch hier wird eine Meldung ausgegeben.

In Zeile 22 wird ein neues Objekt erstellt. Das Zuweisen des Vornamens funktioniert wie gewohnt. In Zeile 27 wird versucht, die Eigenschaft `$alter` zu setzen. Jetzt wird die `__set()`-Methode aufgerufen. In Zeile 29 wird der Versuch unternommen, die nicht-existente Eigenschaft `$name` auszugeben. In diesem Fall wird die `__get()`-Methode aufgerufen.

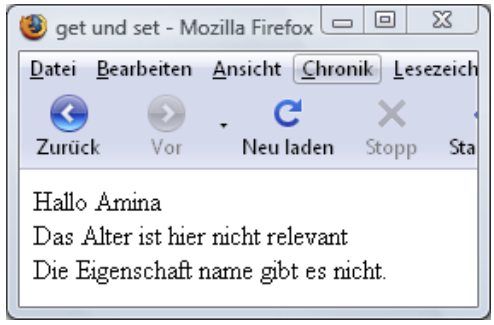


Abbildung 9.19: Die vordefinierten Meldungen beim Zugriff auf die nicht definierten Eigenschaften werden ausgegeben.

Wann ist der Einsatz von `__get()` und `__set()` sinnvoll? Beispielsweise könnten seltener benötigte Eigenschaften einer Klasse in einer Datei ausgelagert werden. Diese wird erst bei Bedarf geladen – über eine `__get()`-Methode, wenn auf eine unbekannte Eigenschaft zugegriffen wird.

9.11.2 __call() und callStatic() – Magie für Methoden

Das, was `__get()` und `__set()` bei Eigenschaften machen, macht `__call()` für Methoden. `__call()` wird – sofern in der Klasse implementiert – aufgerufen, wenn eine nicht definierte Methode eingesetzt wird.

Listing 9.29: Aufruf einer nicht definierten Methode (call.php)

```
01 class Beispiel
02 {
03   public $vorname;
04   public function halloSagen()
05   {
06     echo "Hallo {$this->vorname}";
07   }
08   public function __call($methode, $argumente)
09   {
10     echo "Sie haben $methode aufgerufen. Die gibts nicht.  ";
11    if (!empty($argumente)) {
12       echo "Die übergebenen Argumente: ";
13       foreach ($argumente as $el) {
14         echo "$el ";
15       }
16     }
17   }
18 }
19
20 $bsp = new Beispiel();
21 echo "<br />\n";
22 $bsp->verabschieden("tschüss", "morgen");
```

Im Beispiel sehen Sie wieder die Klasse `Beispiel`. Implementiert ist dieses Mal die Methode `__call()` in Zeile 8. Sie erwartet zwei Parameter: zuerst den Methodennamen und danach die Parameterliste. Auf beide können Sie innerhalb der magischen Methode `__call()` zugreifen. Die Parameterliste (der zweite übergebene Parameter) ist wie zu erwarten ein Array. Ob Parameter übergeben wurden, wird mit `!empty()` überprüft und gegebenenfalls die übergebenen Parameter ausgegeben.

Abbildung 9.20: Beim Aufruf einer Methode, die in der Klasse nicht implementiert ist, wird die vorbereitete Meldung ausgegeben.

PHP erlaubt ab Version 5.3 auch den Aufruf von unbekannten statischen Methoden abzufangen. Hierfür gibt es die magische Methode `__callStatic()`.

Listing 9.30: __callStatic() im Einsatz (callstatic.php)

```
class Beispiel
{
  static function __callStatic($methode, $argumente)
  {
    echo "Sie haben die statische Methode '$methode' aufgerufen. ";
  }
}
Beispiel::etwas();
```

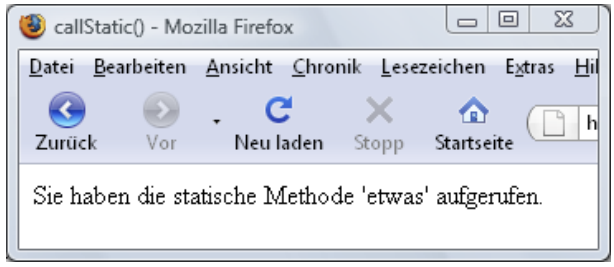


Abbildung 9.21: Ab PHP 5.3 kann auch der Aufruf unbekannter statischer Methoden abgefangen werden.

9.11.3 Dateien automatisch laden über __autoload()

Bei der objektorientierten Programmierung in PHP werden häufig alle Klassen in eigenen Dateien untergebracht, die dann bei Bedarf eingebunden werden. Dieses Laden der externen Dateien kann PHP mithilfe von `__autoload()` für Sie übernehmen.

Im Gegensatz zu den vorher beschriebenen magischen Methoden – wie `__construct()`, `__destruct()`, `__get()`, `__set()`, `__call()` oder auch `__callStatic()` – wird `__autoload()` nicht innerhalb der Klasse definiert, sondern **außerhalb**.

`__autoload()` wird automatisch aufgerufen, wenn ein Objekt einer Klasse erstellt ist, deren Definition nicht vorhanden ist. `__autoload()` erwartet als Parameter den Namen der Klasse.

Nehmen wir einmal an, dass die Datei, in der die Klasse steht, genauso heißt wie die Klasse, nur zusätzlich mit der Endung *.php*. Dann lässt sich `__autoload()` folgendermaßen implementieren:

Listing 9.31: Klassen automatisch laden (autoload.php)

```
function __autoload($klasse) {
  require_once $klasse . ".php";
}
$obj  = new Beispiel1();
```

```
$obj->info();
$obj2 = new Beispiel2();
$obj2->info();
```

Das heißt, wenn ein Objekt von einer nicht geladenen Klasse erstellt wird, wird die Datei mit dem dazugehörigen Namen und der Endung `.php` eingebunden.

Tipp

Hier ist `require_once` die richtige Art der Einbindung – sollte die Einbindung nicht klappen, soll das Skript abgebrochen werden. Außerdem sorgt `require_once` dafür, dass die Datei wirklich nur einmal eingebunden wird. Eine mehrfache Einbindung und damit auch eine mehrfache Klassendefinition würden einen Fehler erzeugen.

Damit Listing 9.31 klappt, müssen natürlich die zwei externen Dateien *Beispiel1.php* und *Beispiel2.php* existieren und die jeweiligen Klassendefinitionen enthalten.

Im Beispiel sind sie einfach gehalten: Sie definieren beide nur eine Methode `info()`, die ausgibt, dass die aktuelle Klasse geladen ist.

Listing 9.32: Die eine geladene Datei (Beispiel1.php)

```
<?php
class Beispiel1
{
  public function info()
  {
    echo get_class($this) . " wurde geladen. ";
  }
}
?>
```

Da die eingebundene Datei *Beispiel1.php* PHP-Code enthält, muss dieser auch in der eingebundenen Datei mit `<?php` eingeleitet werden. Die Datei *Beispiel2.php* sieht genauso aus wie *Beispiel1.php*, bis darauf, dass die Klasse `Beispiel2` heißt.

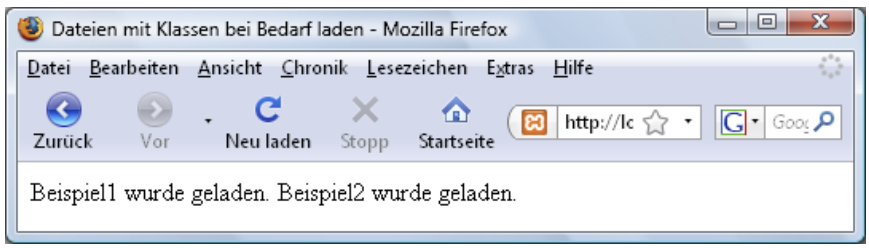


Abbildung 9.22: Die Dateien mit den Klassendefinitionen wurden über __autoload() geladen.

Falls jedoch die Datei nicht existiert, die über `__autoload()` eingebunden wird, erhalten Sie eine entsprechende Fehlermeldung.

9.11.4 Ausgabe steuern über __toString()

Möchten Sie schnell Informationen zu einem Objekt erhalten, können Sie `var_dump()` benutzen. Dieses liefert Ihnen die aktuell gesetzten Eigenschaften. Nehmen wir als Beispiel noch einmal die Klasse `Kunde`.

```
require_once "kunde.php";
$neuerKunde = new Kunde ("Anja");
$neuerKunde->speichern(20);
echo "<pre>";
echo "Infos über var_dump()\n";
var_dump($neuerKunde);
```

```
Infos über var_dump()
object(Kunde)#1 (3) {
  ["name"]=>
  string(4) "Anja"
  ["speicherGesamt"]=>
  int(50)
  ["speicherVerbraucht"]=>
  int(20)
}
```

Abbildung 9.23: Informationen über das Objekt können über var_dump() ausgegeben werden.

Auch `print_r()` liefert brauchbare Ergebnisse:

```
echo "Infos über print_r()\n";
print_r($neuerKunde);
echo "</pre>";
```

```
Infos über print_r()
Kunde Object
(
    [name] => Anja
    [speicherGesamt] => 50
    [speicherVerbraucht] => 20
)
```

Abbildung 9.24: So sehen die per print_r() ausgegebenen Informationen aus.

Schließlich können Sie Objekte wie Arrays in einer `foreach`-Schleife durchlaufen und erhalten alle öffentlichen Eigenschaften und die aktuellen Werte:

Listing 9.33: Informationen über Objekte ausgeben lassen (informationen.php)

```
echo "Objekte lassen sich wie ein Array durchlaufen<br />\n";
foreach($neuerKunde as $k => $v) {
  echo "$k: $v<br />\n";
}
```

```
Objekte lassen sich wie ein Array durchlaufen
name: Anja
speicherGesamt: 50
speicherVerbraucht: 20
```

Abbildung 9.25: Dieses Mal werden Eigenschaften und Werte über eine foreach-Schleife ausgegeben.

Wenn Sie jedoch direkt ein Objekt per `print` oder `echo` ausgeben lassen, liefert dies ein unbefriedigendes Ergebnis. Dies lässt sich mit der magischen Methode `__toString()` ändern. Wenn Sie diese in einer Klasse definieren, können Sie dort bestimmen, was ausgegeben werden soll, wenn `print` oder `echo` bei einem Objekt benutzt wird.

Achtung

Die magische Methode `__toString()` muss einen String zurückliefern.

Listing 9.34: In der Klasse Premiumkunde wird die __toString()-Methode implementiert (tostring.php).

```
01 require_once "kunde.php";
02 class Premiumkunde extends Kunde
03 {
04   public $speicherGesamt = 100;
05   public $farbSchema;
06   public function __construct($name, $speicherVerbraucht = 0, $farbSchema =
       "Sonnenaufgang")
07   {
08     $this->name = $name;
09     $this->speicherVerbraucht = $speicherVerbraucht;
10     $this->farbSchema = $farbSchema;
11   }
12   public function __toString()
13   {
14     $string = "Instanz von Premiumkunde<br />\n"
```

```
15            . "Folgende Eigenschaften sind definiert: <br />\n"
16            . "<ul>\n"
17            . "<li>Name: " . $this->name . "</li>\n"
18            . "<li>speicherVerbraucht: " . $this->speicherVerbraucht . "</li>\n"
19            . "<li>farbSchema: " . $this->farbSchema . "</li>\n"
20            . "</ul>\n";
21      return $string;
22    }
23 }
24
25 $pk = new Premiumkunde("Hans-Heinerich");
26 echo $pk;
```

Im Beispiel wird in der Klasse Premiumkunde in Zeile 12 die `__toString()`-Methode implementiert. In dieser Methode wird ein String mit Informationen zum jeweiligen Objekt zusammengestellt. Die Eigenschaften sollen in Form einer Liste ausgegeben werden. Die `__toString()`-Methode gibt diesen String mit `return` zurück.

In Zeile 26 sehen Sie, dass das Objekt direkt per `echo` ausgegeben wird. Jetzt wird die `__toString`-Methode aufgerufen und gibt das gewünschte Ergebnis aus.

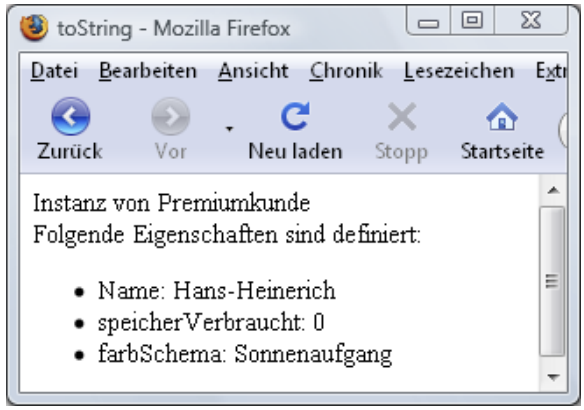


Abbildung 9.26: Mit __toString() lässt sich die Ausgabe beliebig gestalten.

Hinweis

Informationen über Objekte bzw. Klassen erhalten Sie auch über verschiedene von PHP vordefinierte Funktionen:

- `class_exists()` erwartet als Parameter den Namen einer Klasse und gibt `true` zurück, wenn die Klasse existiert, ansonsten `false`.
- `interface_exists()` überprüft entsprechend die Existenz eines Interfaces.

- `get_class_methods()` erwartet als Parameter den Namen einer Klasse und gibt ein Array mit den definierten Methoden zurück.
- `get_class_vars()` erwartet als Parameter den Namen einer Klasse und gibt ein Array mit den vordefinierten Eigenschaften zurück.
- `get_object_vars()` liefert die öffentlichen Eigenschaften eines Objekts.
- `get_class()` zur Ermittlung der Klassennamen eines Objekts haben Sie bereits kennen gelernt (siehe Abschnitt 9.11.2).
- `get_parent_class()` liefert entsprechend zu einem Objekt den Namen der Elternklasse.
- `is_subclass_of()` erwartet zwei Strings als Parameter und gibt an, ob das zuerst genannte Objekt eine Instanz einer Unterklasse der im zweiten Parameter genannten Klasse ist.
- Über den Operator `instanceof` können Sie feststellen, ob ein gegebenes Objekt zu einer bestimmten Klasse gehört.

Eine vollständige Auflistung finden Sie unter *http://php.net/manual/de/ref.classobj.php*.

9.12 Referenzen, Klone und Vergleiche

Eine Besonderheit zeigt sich bei der Zuweisung von Objekten zu Variablen. Sehen wir uns zum Vergleich einmal an, wie es sich bei normalen Variablen verhält. Angenommen, Sie speichern in einer Variablen einen Wert:

```
$a = 5;
```

Und weisen dann die Variable `$a` einer anderen Variablen zu:

```
$b = $a;
```

Dann hat `$b` ebenfalls den Wert 5. Ändern Sie jetzt den Wert von `$a`, so ist `$b` davon nicht betroffen:

```
$a++;
echo $b; /* gibt weiter 5 aus */
```

9.12.1 Referenzen und Klone

Bei Objekten ist dies jedoch anders. Wenn Sie ein Objekt einer anderen Variable zuweisen, kopieren Sie damit nicht das gesamte Objekt, sondern die Variable erhält *nur eine Referenz* auf dieses Objekt.

```
01 class Beispiel
02 {
03   public $farbe;
04   public function info()
05   {
06     echo "Farbe ist {$this->farbe}<br />\n";
07   }
08 }
09 $obj1 = new Beispiel();
10
11 $obj2 = $obj1;
12 $obj1->farbe = "blau";
13 $obj1->info();
14 $obj2->info();
15
16 $obj1->farbe = "orange";
17 $obj2->info();
```

Im Listing wird eine einfache Klasse mit Namen `Beispiel` erzeugt. Sie hat die Eigenschaft `$farbe` und eine Methode `info()`, die die aktuelle Farbe ausgibt.

In Zeile 9 wird eine Instanz der Klasse `Beispiel` erzeugt. In Zeile 11 wird dieses Objekt der Variable `$obj2` zugewiesen. Schließlich wird in Zeile 12 die Eigenschaft `$farbe` des Objekts `$obj1` auf `blau` gesetzt. Für beide Objekte wird die Methode `info()` aufgerufen (Zeile 13 und 14): Da `$obj2` nur eine Referenz auf `$obj1` enthält, ist das Ergebnis dasselbe. Beides Mal wird ausgegeben: »Farbe ist blau".

Das zeigen auch noch einmal Zeilen 16 und 17. In Zeile 16 wird die Eigenschaft `$farbe` von `$obj1` auf den Wert `orange` gesetzt. Liest man danach diese Eigenschaft über die `info()`-Methode aus, zeigt sich, dass auch bei `$obj2` die Eigenschaft `$farbe` den Wert `orange` hat. Da `$obj2` nur eine Referenz auf `$obj1` enthält, betreffen alle Änderungen, die Sie an `$obj1` durchführen genauso `$ob2`. Es sind einfach zwei verschiedene Namen für dasselbe Objekt.

Falls Sie jedoch nicht eine Referenz auf ein Objekt erstellen möchten, sondern ein Objekt mit allen Eigenschaften kopieren möchten, so dass Sie dann zwei Objekte haben, die sich unabhängig voneinander verändern lassen, müssen Sie ein Objekt über `clone` klonen.

Erweitern wir dafür einmal das Beispiel von eben. Die Klassendefinition bleibt wie gehabt.

Listing 9.35: Referenzen und Klone (referenzen_klone.php)

```
09 $obj1 = new Beispiel();
10
11 $obj2 = $obj1;
12 $obj1->farbe = "blau";
13 $obj1->info();
```

```
14 $obj2->info();
15
16 $obj3 = clone $obj1;
17 $obj1->farbe = "orange";
18 $obj3->info();
19 $obj2->info();
```

In Zeile 16 wird `$obj3` als Klon von `$obj1` mit allen Eigenschaften erstellt. Wird jedoch nach dem Klonvorgang `$obj1` verändert – in Zeile 17 wird die Eigenschaft `$farbe` auf `orange` gesetzt – so betrifft dies `$obj3` nicht mehr. Die Zeile 18 erzeugt die Ausgabe »Farbe ist blau« – denn das entspricht dem Wert, den die Eigenschaft `$farbe` in dem Moment hatte, als `$obj3` als Klon von `$obj1` erstellt wurde.

Abbildung 9.27: Ausgabe von Listing 9.35

Hinweis

In PHP 4 war dies genau umgekehrt: Standardmäßig haben Sie in PHP 4 durch die Zuweisung ein Objekt geklont. Um das zu verhindern, mussten Sie das `&`-Zeichen einsetzen.

Soll beim Klonen mehr geschehen, müssen Sie die magische Methode `__clone()` in Ihrer Klasse implementieren. Sie wird automatisch aufgerufen, wenn ein Objekt geklont wird. Hier könnten Sie beispielsweise einzelne Eigenschaften wieder zurücksetzen, bei denen ein anderer Anfangswert wichtig ist.

9.12.2 Objekte vergleichen

Zum Vergleichen von Objekten können Sie `==` und `===` verwenden. Diese verhalten sich jedoch bei Objekten anders als bei normalen Variablen. `==` überprüft, ob die Eigenschaften identisch sind, `===` überprüft hingegen, ob der Objekt-Handler identisch ist, also auf dasselbe Objekt verweist.

Ein Beispiel demonstriert die Unterschiede:

Listing 9.36: Referenzen und Klone vergleichen (objekte_vergleichen.php)

```
01 class Beispiel
02 {
03   public $farbe;
04   public function info()
05   {
06     echo "Farbe ist {$this->farbe}<br />\n";
07   }
08 }
09 $obj1 = new Beispiel();
10 $obj1->farbe = "blau";
11
12 $obj2 = $obj1;
13 $obj3 = clone $obj1;
14
15 echo "<pre>";
16 var_dump($obj1 == $obj2);
17 var_dump($obj1 === $obj2);
18
19 var_dump($obj1 == $obj3);
20 var_dump($obj1 === $obj3);
21 echo "</pre>";
```

Im Listing wird wieder eine einfache Beispielklasse mit der Eigenschaft `$farbe` benutzt. In Zeile 9 wird ein neues Objekt instanziiert und die Eigenschaft `$farbe` auf `blau` gesetzt.

In Zeile 12 wird eine Referenz auf `$obj1` erstellt und in `$obj2` gespeichert. `$obj1` und `$obj2` verweisen so auf dasselbe Objekt.

In Zeile 13 wird mit `clone` eine Kopie von `$obj1` erstellt und in der Variable `$obj3` gespeichert. `$obj3` hat damit dieselben Eigenschaften wie `$obj1`, ist aber ein eigenständiges Objekt.

Dann werden ab Zeile 16 mehrere Vergleiche durchgeführt und das Ergebnis mit `var_dump()` direkt ausgegeben. Wie zu erwarten, ergibt `$obj1 == $obj2` und `$obj1 === $obj2` den Booleschen Wert `true`: Beide verweisen schließlich auf dasselbe Objekt.

Auch `$obj == $obj3` ergibt `true`, da die Eigenschaften der beiden Objekte übereinstimmen. Allerdings sind sie verschiedene Objekte, deswegen ergibt `$obj1 === $obj3` den Booleschen Wert `false`.

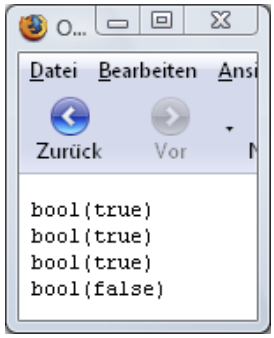


Abbildung 9.28: Das Ergebnis der Vergleiche

9.13 Namensräume

Bei großen Projekten, die nicht objektorientiert programmiert sind, kann es leicht zu Namenskonflikten kommen, wenn es mehrere Funktionen mit einem vorgegebenen Namen gibt, also beispielsweise zwei Funktionen mit dem Namen `starten()`. Diese Art von Konflikten ist bei der objektorientierten Programmierung weniger wahrscheinlich, weil die Funktionen – dann als Methoden – ja an die jeweiligen Klassen gebunden sind. Zwei Methoden mit dem Namen `starten()`, die bei unterschiedlichen Klassen definiert sind, stören sich nicht. Was ist aber, wenn bei einem Projekt mehrere externe Bibliotheken zum Einsatz kommen, die zwei gleichnamige Klassen haben? Denkbar ist das immer bei so generischen Klassen wie `User` oder Ähnlichem. Genau an dieser Stelle kommen die in PHP 5.3 neu eingeführten Namensräume ins Spiel – die neben Klassen auch bei Funktionen und Konstanten relevant sind.

Tipp

Das PHP-Manual vergleicht Namensräume mit Ordnern im Dateisystem. Innerhalb von zwei verschiedenen Ordnern können Dateien desselben Namens vorkommen, ohne dass sich diese in die Quere kommen.

9.13.1 Grundlegendes

Einen Namensraum geben Sie hinter dem Schlüsselwort `namespace` an. Danach können Sie die zu dem Namensraum gehörigen Klassen, Funktionen und Konstanten definieren.

```
namespace meinprojekt;
class Benutzer {}
function wastun() {
  echo "getan";
}
const zahl = 42;
```

Achtung

Die `namespace`-Anweisung muss sich im Code zualleroberst befinden, es dürfen sich keine PHP-Befehle (mit Ausnahme von `declare()` zur Angabe einer Kodierung) darüber befinden, auch darf kein HTML-Code davor ausgegeben werden. Ansonsten erhalten Sie die in Abbildung 9.29 gezeigte Fehlermeldung.

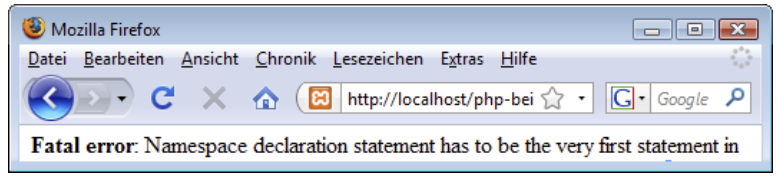


Abbildung 9.29: Wenn vor der namespace-Anweisung etwas anderes steht, erzeugt das einen fatalen Fehler.

Innerhalb der Datei selbst, in der Sie den Namensraum deklariert haben, können Sie dann die Klassen/Funktionen/Konstanten direkt benutzen, weil Sie sich ja innerhalb dieses Namensraums befinden:

Listing 9.37: Namensraum definieren und dazugehörige Klassen, Funktionen und Konstanten einsetzen (namensraum.php)

```
namespace meinprojekt;
class Benutzer {}
function wastun() {
  echo "getan";
}
const zahl = 42;
wastun();
$a = new Benutzer();
echo " " . zahl;
```

Tipp

Über die vordefinierte Konstante `__NAMESPACE__` haben Sie Zugriff auf den aktuell benutzten Namensraum.

Wenn Sie hingegen die Datei *namensraum.php* in eine andere Datei einbinden und hierin auf eine Klasse, Funktion oder Konstante zugreifen möchten, so müssen Sie den Namensraum davor angeben. Dafür geben Sie zuerst den Namensraum, dann einen Backslash als Separator und schließlich die Funktion/Klasse/Konstante an:

Listing 9.38: Die Funktion wastun() aus dem Namensraum meinprojekt wird benutzt (namensraum_benutzen.php).

```
require "namensraum.php";
/* das geht nicht: */
// wastun();
/* das hingegen geht */
meinprojekt\wastun();
```

Bei der direkten Angabe von `wastun()` erhalten Sie hingegen eine Fehlermeldung, dass die entsprechende Funktion nicht gefunden wurde.

Hinweis

Die Namen von Namensräumen können selbst auch den Backslash beinhalten, so dass sich diese noch besser organisieren lassen. Eine Gruppe von Framework- und Bibliotheksentwicklern hat im Mai 2009 beschlossen, ihre Namensräume so aufzubauen, dass zuerst der Name des Herstellers und danach der Name des Pakets angegeben wird (*http://news.php.net/php.standards/2*). Dadurch ist bei der Verwendung immer klar, worum es sich handelt und gleichnamige Pakete kommen sich nicht in die Quere:

```
namespace pear2\text_diff;
namespace zend\controller;
```

9.13.2 Absolut und relativ

Selbstverständlich können Sie in mehreren Dateien denselben Namensraum definieren. Aber auch das Umgekehrte ist möglich: die Definition von mehreren Namensräumen innerhalb einer Datei. Dafür geben Sie einfach die Namensräume untereinander an:

```
namespace Eins;
const zahl = 1;
namespace Zwei;
```

Nehmen wir jetzt an, Sie möchten innerhalb des Namensraums `Zwei` die Konstante `zahl` aufrufen.

Die erste Möglichkeit, die fehlschlägt, ist den Namen der Konstante direkt anzugeben:

```
namespace Zwei;
echo zahl;
```

Das kann nicht funktionieren, da `zahl` ja im aktuellen Namensraum nicht definiert ist.

Ein zweiter Ansatz könnte so aussehen, dass man den Namensraum folgendermaßen angibt:

```
namespace Zwei;
echo Eins\zahl;
```

Aber auch das funktioniert nicht, denn `Eins\zahl` ist eine relative Angabe und wird damit *im Verhältnis zum aktuellen Namensraum* `Zwei` aufgelöst. So wird nach der Konstante `Zwei\Eins\zahl` gesucht, die es nicht gibt.

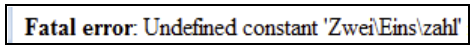


Abbildung 9.30: Die Fehlermeldung zeigt deutlich, dass Eins\zahl innerhalb des Namensraums Zwei als Zwei\Eins\zahl aufgelöst wird.

Um innerhalb des Namensraums `Zwei` auf ein Element des Namensraums `Eins` zuzugreifen, müssen Sie eine *absolute Angabe* wählen, indem Sie vor der Namensraumangabe einen Backslash schreiben:

```
namespace Zwei;
echo \Eins\zahl;
```

Hier sehen Sie noch einmal das Beispiel in seiner Gesamtheit:

Listing 9.39: Einsatz mehrerer Namensräume in einer Datei (mehrere_namensraeume.php)

```
namespace Eins;
const zahl = 1;

namespace Zwei;
/* geht nicht, zahl ist im aktuellen Namensraum nicht definiert */
// echo zahl;
/* geht nicht, da Eins\zahl in Zwei\Eins\zahl aufgelöst wird */
//echo Eins\zahl;
/* so geht es: */
echo \Eins\zahl;
```

Diese Art von absoluten und relativen Angaben erinnern natürlich deutlich an die Pfadangaben im Dateisystem und man kann sie ganz ähnlich sehen. Auf die absoluten Angaben werden wir gleich noch mal zu sprechen kommen.

Es gibt eine alternative Syntax für die Verwendung von mehreren Namensräumen innerhalb einer Datei: Sie können den zum jeweiligen Namensraum gehörenden Code auch in geschweifte Klammern einfassen:

Listing 9.40: Zum Namensraum gehöriger Code kann in geschweifte Klammern eingefasst werden (mehrere_namensraeume_alternativ.php).

```
namespace Eins {
  const zahl = 1;
}
namespace Zwei {
  echo \Eins\zahl;
}
```

Wichtig ist aber dann, dass Sie keinen Code außerhalb der Namensraum-Angaben, also außerhalb der geschweiften Klammern notieren, sonst erhalten Sie die in Abbildung 9.31 gezeigte Fehlermeldung.

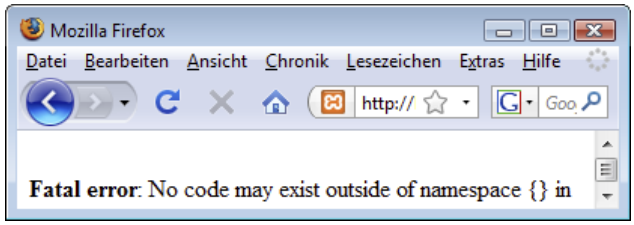


Abbildung 9.31: Kein Code ist außerhalb der geschweiften Klammern der namespace-Deklarationen erlaubt.

9.13.3 Abkürzungen: use benutzen

Wenn die Namensräume lang werden, kann das viel Schreibarbeit bedeuten. Verkürzungen sind über `use` möglich.

Nehmen wir einmal folgende »Bibliothek« an – die einen Namensraum definiert und hier der Einfachheit halber nur eine Konstante:

Listing 9.41: Einfache Bibliothek (bibliothek.php)

```
namespace anwendung\bibliothek;
const zahl = 42;
```

Wenn Sie jetzt außerhalb dieser Datei auf die Konstante zugreifen, müssen Sie den Namensraum angeben, also beispielsweise `anwendung\bibliothek\zahl` schreiben. Kürzer geht es über `use`:

Listing 9.42: Abkürzungen über use (usenutzen.php)

```
require_once "bibliothek.php";
use anwendung\bibliothek as a;
echo a\zahl;
```

Mit `use` wird `a` als Alias für den Namensraum definiert, so dass Sie über `a\zahl` auf die Konstante zugreifen können.

Außerdem können Sie mit `use` auch einen Alias für Klassen definieren. Hierfür brauchen wir erst einmal eine einfache Klassendefinition innerhalb eines Namensraums:

Listing 9.43: Definition der Klasse innerhalb des Namensraums meinprojekt (namensraum_beispiel.php)

```
namespace meinprojekt;
class Benutzer {
  public function __construct() {
     echo "Benutzer erstellt";
  }
}
```

Jetzt soll die Klasse in einer anderen Datei verwendet werden, allerdings soll nicht immer der vollständige Name benutzt werden. Dafür ist `use` da:

Listing 9.44: use zur Vereinfachung des Zugriffs auf Klassen (namensraum_use.php)

```
require "namensraum_beispiel.php";

use meinprojekt\Benutzer;
$a = new Benutzer();
```

Nach der Angabe von `use meinprojekt\Benutzer` können Sie die Klasse direkt als `Benutzer()` verwenden. Das funktioniert allerdings nur bei Klassen, nicht jedoch bei Funktionen oder Konstanten.

Das oben angegebene `use meinprojekt\Benutzer` ist eigentlich eine Abkürzung für:

```
use meinprojekt\Benutzer as Benutzer;
```

Und das heißt natürlich, Sie können mit `as` beliebige Aliasnamen für Ihre Klasse vergeben:

Listing 9.45: Über use können Sie Aliasnamen für Klassen vergeben (namensraum_use_2.php).

```
require "namensraum_beispiel.php";

use meinprojekt\Benutzer as B;
$a = new B();
```

9.13.4 Globaler Namensraum

Der globale Namensraum ist der namenslose Namensraum, in dem sich beispielsweise alle von PHP vordefinierten Funktionen wie `htmlspecialchars()`, `date()` usw. usf. befinden.

Hinweis

Die magische Konstante `__NAMESPACE__` ist beim globalen Namensraum leer.

Wenn Sie innerhalb eines Namensraums ein Element aus dem globalen Kontext benutzen wollen, schreiben Sie einen Backslash an den Anfang:

```
namespace Eins;
/* ruft date() aus dem globalen Kontext auf */
$b = \date(...);
```

Diesbezüglich gibt es aber einen Unterschied zwischen Klassen auf der einen Seite und Funktionen und Konstanten auf der anderen Seite. Dieser zeigt sich, wenn ein Element im aktuellen Namensraum *nicht gefunden wird*: Dann wird nämlich bei Funktionen und Konstanten im globalen Namensraum gesucht, ob eine entsprechende Funktion/Konstante dort existiert. Bei Klassen passiert das nicht. Wenn Sie eine Klasse aus dem globalen Namensraum benutzen wollen, so müssen Sie, wenn Sie innerhalb eines anderen Namensraums sind, vor dem Klassennamen *immer* einen Backslash schreiben.

Das heißt, Sie könnten sofern es im Namensraum `Eins` keine Funktion mit Namen `date()` gibt, ebenfalls folgenden Code nutzen, um `date()` aus dem globalen Kontext aufzurufen:

```
namespace Eins;
/* ruft date() aus dem globalen Kontext auf, wenn date() im aktuellen Namensraum
nicht definiert ist */
$b = date(...);
```

9.14 Fehlerbehandlung mit der Exception-Klasse

Bisher ging es darum, wie Sie selbst Klassen definieren und nutzen. Daneben gibt es auch von PHP vordefinierte Klassen, und besonders praktisch ist die `Exception`-Klasse.

Exceptions (»Ausnahmen«) ermöglichen eine gezielte Fehlerbehandlung. Bei einem Fehler wird das Programm nicht sofort beendet, sondern der Fehler wird der aufrufenden Funktion mitgeteilt, die dann den Fehler behandeln kann. Zum Einsatz von Exceptions erstellen Sie neue Instanzen der von PHP vorgegebenen Klasse `Exception`.

Der Teil des Programms, der Probleme machen kann, wird dabei in einem `try`-Block notiert. Wenn ein Fehler auftritt, wird dieser durch den nachfolgenden `catch`-Block abgefangen.

Sehen wir uns das einmal am Beispiel an. Im Listing gibt es eine Funktion namens `teilen()`, die das Ergebnis der Division von 1 durch den übergebenen Parameter liefert.

```
function teilen($x)
{
  return 1/$x;
}
```

Ist jedoch der übergebene Wert 0, so würde es zu einem Fehler kommen. In diesem Fall wird in der verbesserten Version des Listings über `throw new Exception()` eine Ausnahme ausgelöst. Damit wird ein neues Objekt der `Exception`-Klasse erstellt. Genaueres zu dieser Klasse etwas später.

```
01 function teilen($x)
02{
03   if($x==0) {
04     throw new Exception("Teilen durch 0! ");
05   } else {
06     return 1/$x;
07   }
08 }
```

Hinweis

Wenn Sie sich hingegen in einer Datei befinden, die einen Namensraum deklariert, müssen Sie zum Zugriff auf die globale Exception-Klasse vor `Exception` einen Backslash schreiben:

```
throw new \Exception("Teilen durch 0! ");
```

Nun kommt der zweite Teil des Listings, in dem diese Funktion `teilen()` mit unterschiedlichen Parametern aufgerufen wird. Diese Funktionsaufrufe sind in einem `try`-Block gekapselt. Darauf folgt ein `catch`-Block mit der Fehlerbehandlung: Im Beispiel wird eine Meldung ausgegeben.

Listing 9.46: Fehlerbehandlung über Exception (exception.php)

```
09 try {
10   echo teilen(4) . "<br />\n";
11   echo teilen(0) . "<br />\n";
12   echo teilen(5) . "<br />\n";
13 } catch (Exception $e) {
14   echo "Exception gefangen: " . $e->getMessage();
15 }
```

Die Funktion `teilen()` wird dreimal aufgerufen. Der erste Aufruf (Zeile 10) verläuft normal. Beim zweiten Aufruf in Zeile 11 wird hingegen `0` als Parameter übergeben und damit die `Exception` ausgelöst. Wenn eine `Exception` ausgelöst wird, springt der PHP-Interpreter zum folgenden `catch`-Block (Zeile 13). Der Code nach der `Exception` und vor dem `catch`-Block (Zeile 12) wird hingegen nicht mehr ausgeführt.

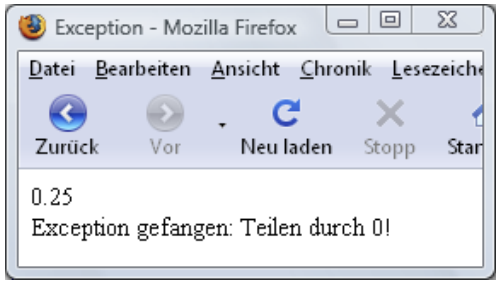


Abbildung 9.32: Der Fehler ist abgefangen.

Bei der Instanziierung des neuen `Exception`-Objekts in Zeile 4 wird ein Text übergeben. Dieser beinhaltet die eigentliche Fehlermeldung, die Sie dann über die Methode `getMessage()` (Zeile 14) ausgeben lassen können.

Die Fehlerbehandlung kann differenzierter gestaltet werden, indem Sie je nach Situation unterschiedliche Exceptions »werfen« und dabei einen Fehlercode übergeben. Beim Fangen der `Exception` kann je nach Fehlercode unterschiedlich reagiert werden. Auf den Fehlercode können Sie über die Methode `getCode()` der `Exception` zugreifen.

Listing 9.47: Exceptions mit unterschiedlichen Fehlercodes (exception_erweitert.php)

```
01 function teilen($x)
02 {
03   if (!is_numeric($x)){
04     throw new Exception("keine Zahl", 1);
05   } elseif ($x == 0)  {
06      throw new Exception("Teilen durch 0! ", 2);
07   } else {
08     return 1/$x;
09   }
```

```
10 }
11
12 try {
13   echo teilen(4) . "<br />\n";
14   echo teilen("hallo") . "<br />\n";
15 } catch (Exception $e) {
16   if ($e->getCode() == 2) {
17     echo "Falscher Wert: " . $e->getMessage();
18   } elseif ($e->getCode() == 1) {
19     echo "Falscher Datentyp: " . $e->getMessage();
20   }
21 }
```

Wieder sehen Sie in Listing 9.47 die Funktion `teilen()`. Dieses Mal wird unterschieden: Ist der übergebene Parameter nicht numerisch, wird eine `Exception` mit dem Fehlercode 1 geworfen, ist er 0, hingegen eine `Exception` mit dem Fehlercode 2.

In Zeile 16 wird beim Fangen der `Exception` unterschieden: Wenn der Fehlercode, der über `$e->getCode()` ermittelt wird, den Wert 2 hat, wird eine andere Meldung ausgegeben, als wenn er den Wert 1 hat (Zeile 18). Selbstverständlich lassen sich hier auch beliebige andere Reaktionen angeben – auch beispielsweise ein Abbruch des Skripts über `exit`.

Tipp

Ein weiteres Beispiel für den Einsatz von Exceptions zur Fehlerbehandlung zeigt Kapitel 12.

Die in PHP vordefinierte `Exception`-Klasse, die bisher zum Einsatz kam, kann auch erweitert werden. Sie hat folgende Eigenschaften und Methoden:

```
01 class Exception   {
02   /* Eigenschaften */
03   protected string $message ;
04   private string $string ;
05   protected int $code ;
06   protected string $file ;
07   protected int $line ;
08   private array $trace ;
09   /* Methoden */
10   public __construct ([ string $message [, int $code ]] )
11   final public string getMessage ( void )
12   final public int getCode ( void )
13   final public string getFile ( void )
```

```
14   final public string getLine ( void )
15   final public array getTrace ( void )
16   final public string getTraceAsString ( void )
17   public string __toString ( void )
18   final private string __clone ( void )
19 }
```

Erst zu den Eigenschaften:

- `$message`: die Meldung, die im Fehlerfall ausgegeben wird
- `$string`: der interne Name der Exception
- `$code`: die Fehlernummer der Exception
- `$file`: Name der Datei, in der die Exception aufgetreten ist
- `$line`: Nummer der Zeile, in der die Exception aufgetreten ist
- `$trace`: der Stacktrace – Informationen zum Fehler als Array

Nun zu den Methoden:

- In Zeile 10 steht die Konstruktormethode. Fakultativ können ein Fehlertext und eine Fehlernummer übergeben werden. Der Standardfehlertext ist ein leerer String und die Standardfehlernummer 0.

Die anderen Methoden liefern die angegebenen Informationen:

- `getMessage()` liefert die Fehlermeldung
- `getCode()` die Fehlernummer
- `getFile()` die Datei, in der der Fehler aufgetreten ist
- `getLine()` die Zeile
- `getTrace()` den Stacktrace, d. h. Infos über Datei, Zeile und eventuell Funktion etc. als Array
- `getTraceAsString()` dieselben Infos wie `getTrace()`, aber direkt als String zur Ausgabe

9.15 Überblick über die bei der objektorientierten Programmierung benutzten Schlüsselwörter

Es gibt viele Schlüsselwörter/Operatoren/Methoden, die bei der objektorientierten Programmierung relevant sind. Die folgende Tabelle listet sie alphabetisch auf – mit kurzer Erklärung und Verweis auf den entsprechenden Abschnitt, in dem sie behandelt werden.

Schlüsselwort	Funktion	Abschnitt
`->`	Zugriff auf Eigenschaften und Methoden	9.1
`::`	Doppeldoppelpunkt-Operator, auch Gültigkeitsbereichsoperator oder Paamayim Nekudotayim genannt	9.4, 9.13
`abstract`	Eine Methode, die als `abstract` deklariert wird, muss in einer abgeleiteten Klasse überschrieben werden.	9.8.2
`__autoload()`	Magische Methode, die automatisch aufgerufen wird, wenn ein Objekt einer Klasse erstellt wird, deren Definition nicht vorhanden ist.	9.11.3
`__call()`	Magische Methode, die automatisch aufgerufen wird, wenn eine in einer Klasse nicht definierte Methode eingesetzt wird.	9.11.2
`__callStatic()`	Magische Methode, die automatisch aufgerufen wird, wenn eine in einer Klasse nicht definierte statische Methode eingesetzt wird.	9.11.2
`clone`	Erstellt eine exakte Kopie eines Objekts, die dann unabhängig vom ursprünglichen Objekt verändert werden kann.	9.12
`__clone()`	Magische Methode, die automatisch aufgerufen wird, wenn ein Objekt geklont wird.	9.12
`const`	Zur Definition von Konstanten in Klassen	9.4
`__construct()`	Methode, die automatisch aufgerufen wird, wenn ein neues Objekt erstellt wird.	9.2
`__destruct()`	Methode, die automatisch aufgerufen wird, wenn ein neues Objekt zerstört wird.	9.2
`extends`	Relevant bei der Vererbung zur Ableitung einer Klasse von der Basisklasse.	9.6
`final`	Eine als `final` definierte Methode kann in einer abgeleiteten Klasse nicht überschrieben werden.	9.8.1
`__get()`	Magische Methode, die automatisch aufgerufen wird, wenn versucht wird, eine nicht definierte Eigenschaft auszulesen.	9.11.1
`implements`	Eine Schnittstelle wird über `implements` implementiert, d. h. die Methodendefinitionen mit Inhalten gefüllt.	9.8.3
`interface`	Eine Schnittstelle enthält nur öffentliche Eigenschaften, Konstanten und Methoden, wobei die Methoden noch nicht implementiert sind.	9.8.3
`namespace`	Zur Definition eines Namensraums	9.13
`parent`	Verweis auf die Elternklasse	9.6

Tabelle 9.1: Überblick über die bei der Objektorientierung verwendeten Schlüsselwörter, Operatoren, magische Methoden und mehr

Schlüsselwort	Funktion	Abschnitt
`private`	Regelt den Zugriff auf Eigenschaften und Methoden: Auf als `private` gekennzeichnete Methoden/Eigenschaften kann nur innerhalb der Klasse selbst zugegriffen werden, nicht in einer abgeleiteten Klasse oder über ein Objekt.	9.7
`protected`	Regelt den Zugriff auf Eigenschaften und Methoden: Auf als `protected` gekennzeichnete Methoden/Eigenschaften kann innerhalb der Klasse selbst und in einer abgeleiteten Klasse zugegriffen werden, nicht über ein Objekt.	9.7
`public`	Regelt den Zugriff auf Eigenschaften und Methoden: Auf als `public` gekennzeichnete Methoden/Eigenschaften kann von überall zugegriffen werden, sie sind öffentlich.	9.7
`self`	Verweis auf die Klasse selbst	9.4, 9.10, 9.10.3
`__set()`	Magische Methode, die automatisch aufgerufen wird, wenn versucht wird, eine nicht definierte Eigenschaft zu setzen.	9.11.1
`static`	Kann bei Eigenschaften oder Methoden eingesetzt werden. Diese können dann direkt ohne Initialisierung eines neuen Objekts aufgerufen werden.	9.10
`static`	`static` kann anstelle von `self` zum Zugriff auf statische Methoden innerhalb einer Klassendefinition benutzt werden. Es wird später, d. h. nicht zur Kompilierzeit, sondern zur Laufzeit durch den Namen der aufrufenden Klasse ersetzt (Late Static Binding).	9.10.3
`$this`	Verweis auf das Objekt selbst	9.2
`__toString()`	Magische Methode, die automatisch aufgerufen wird, wenn `print` oder `echo` bei einem Objekt benutzt wird. Damit können Sie bestimmen, welche Informationen dann ausgegeben werden sollen.	9.11.4
`use`	Dient dem Import von Namensräumen in den aktuellen Kontext.	9.13

Tabelle 9.1: Überblick über die bei der Objektorientierung verwendeten Schlüsselwörter, Operatoren, magische Methoden und mehr (Forts.)

10 Daten komfortabel verwalten mit MySQL

MySQL ist eines der am häufigsten in Verbindung mit PHP eingesetzten Datenbankmanagementsysteme. In diesem Kapitel erfahren Sie Grundlegendes zu MySQL: Sie sehen einerseits, wie Sie das kostenlose Tool phpMyAdmin zur Verwaltung von MySQL-Datenbanken nutzen, lernen aber gleichzeitig die wichtigsten MySQL-Befehle selbst zu schreiben.

Ein Datenbankmanagementsystem wie MySQL brauchen Sie zur Verwaltung Ihrer Daten, das heißt um sie zu speichern, zu ändern oder auch zu löschen. Bei Daten kann es sich um alles Mögliche handeln – sei es, dass Sie die Kommentare zu einem Beitrag speichern möchten, umfangreiche Termine, Neuigkeiten, Benutzer mit Kennwörtern, Adressen etc.

Solange es nicht zu viele Daten sind, lassen sich diese auch in assoziativen Arrays speichern. Assoziative Arrays werden aber bei komplexen Daten schnell unhandlich – und Sie möchten meistens auch Änderungen an diesen Daten vornehmen und speichern – und das ist auf dem gängigen Weg mit Arrays nicht möglich. Daten speichern können Sie natürlich auch mit Textdateien (siehe Kapitel 12). Aber wesentlich komfortabler und besser zu verwalten sind Daten, die in einer Datenbank gespeichert sind. Zwei weitere Gründe, die für die Benutzung einer Datenbank sprechen, sind, dass sich zum einen Datenbanken ganz wunderbar über PHP ansprechen lassen und dass zum anderen eine Datenbank bei den meisten Hostingangeboten mit PHP auch schon dazugehört.

10.1 MySQL und mehr

Es gibt viele weitere Datenbankmanagementsysteme neben dem äußerst beliebten MySQL. Und PHP kann auch mit vielen anderen Datenbankmanagementsystemen zusammenarbeiten. Es gibt Erweiterungen für dBase, DB++, FrontBase, filePro, Firebird/InterBase, Informix, IBM DB2, Ingres II, MaxDB, mSQL, Mssql (Microsoft SQL Server), OCI8 (Oracle OCI8), Ovrimos SQL, Paradox (Paradox File Access), PostgreSQL und Sybase. Die Erweiterungen finden Sie im Manual unter *http://www.php.net/manual/de/refs.database.php*.

Hinweis

Eine Sonderrolle nimmt SQLite ein. Wenn in PHP die Unterstützung für SQLite installiert ist, können Sie die Datenbankfunktionalität von SQLite nutzen, ohne dass Sie zusätzliche Software benötigen, d.h., Sie benötigen keinen Datenbankserver. Die Datenbank selbst wird in einer einzigen Datei gespeichert.

SQLite ist gut geeignet, wenn Sie häufig Inhalte aus einer Datenbank auslesen und nur sporadisch neue ergänzen. Werden hingegen häufig Daten in die Datenbank geschrieben, ist SQLite nicht so effizient.

PHP 5.3 bietet standardmäßig Unterstützung für SQLite in der Version 3. Die Erweiterung `SQLite3` bietet einen objektorientierten Zugriff und attraktive Features wie *Prepared Statements* (mehr dazu in Kapitel 11).

PHP bietet neben diesen Erweiterungen zur Zusammenarbeit mit den einzelnen Datenbankmanagementsystemen auch mehrere Abstraktionsebenen für Datenbanksysteme. Damit können Sie Code schreiben, der unabhängig vom eingesetzten Datenbankmanagementsystem funktioniert (*http://www.php.net/manual/de/refs.database.abstract.php*).

Im Folgenden wird gezeigt, wie Sie mit MySQL arbeiten, das ja bei den meisten Hostingpaketen im Internet mit PHP gebündelt ist und beispielsweise auch bei XAMPP automatisch mitinstalliert wird. Viele bekannte Open Source-Projekte wie das Content Management System TYPO3 oder die Blogging-Software Wordpress basieren ebenfalls auf PHP&MySQL. Falls Sie später einmal mit einem anderen Datenbankmanagementsystem arbeiten, heißen die einzelnen benötigten PHP-Befehle zwar anders, aber das Grundprinzip ist dasselbe.

MySQL ist dual lizenziert. Zum einen ist es eine freie Software, die unter der General Public License (GPL) steht. MySQL kann aber auch mit einer kommerziellen Lizenz eingesetzt werden. Die kommerzielle Lizenz benötigen Sie, wenn Sie kommerzielle Software entwickeln und MySQL (die Software) mit Ihren Produkten ausliefern, aber gleichzeitig nicht den Quellcode unter GPL publizieren. Genauere Informationen zur Lizenz lesen Sie unter *http://www.mysql.com/about/legal/licensing/*.

Sofern Sie nicht ein Komplettpaket wie XAMPP einsetzen, sondern sich bisher nur PHP und einen Webserver installiert haben, müssen Sie MySQL zusätzlich installieren. Wie das geht, erfahren Sie unter *http://dev.mysql.com/doc/refman/5.1/de/installing.html*. Beim Provider ist das üblicherweise schon für Sie erledigt.

Unter XAMPP müssen Sie kontrollieren, ob der MySQL-Server gestartet ist und das sonst bei Bedarf nachholen (siehe Kapitel 2).

10.2 Datenbanken – Grundlegendes

Datenbanksysteme dienen der elektronischen Datenverwaltung. Sie bestehen einerseits aus dem Datenbankverwaltungssystem und andererseits den eigentlichen Datenbanken mit den Daten. Am häufigsten eingesetzt werden *relationale* Datenbanksysteme. Das bedeutet, dass die einzelnen Daten in Tabellen gespeichert werden und die Tabellen in Relationen zueinander stehen.

In einer Tabelle könnten beispielsweise Produkte mit ihren Preisen gespeichert werden:

pfl_id	name	beschreibung	preis
1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum...	7.00
2	Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birke...	8.50
3	Pfeifenwinde	große, dachziegelartig übereinander liegenden Blät...	11.00
4	Filigranfarn	feine mehrfachgefiederte Wedel	4.00
5	Kahle Apfelbeere	Wildobst für viele Standorte	5.00
6	Essigrose	*NULL*	11.00
7	Büffelbeere	Gattung der Ölweidengewächse	2.00
8	Hopfenbuche	Pflanzengattung aus der Familie der Birkengewächse	6.00
9	Schmetterlingsstrauch	schöne Blüten	6.00

Abbildung 10.1: Tabelle mit Produkten

Tabellen bestehen aus Zeilen (*rows*) und Spalten (*columns*). Eine Zeile einer Tabelle nennt man auch einen Datensatz (*record*). `7 - Büffelbeere - Gattung der Ölweidengewächse - 2.00` ist z. B. ein Datensatz.

Statt von Spalten spricht man auch von Feldern (*fields*). Beim Erstellen einer Tabelle definieren Sie, wie viele Felder die Tabelle enthalten soll und legen den Inhaltstyp der Felder fest. `preis` enthält beispielsweise nur Zahlen, `beschreibung` hingegen Zeichenketten. In SQL können Sie hier genaue Angaben über die Länge des Inhalts machen – welcher Art und wie groß beispielsweise die Zahlen sein können, die in `preis` abgespeichert werden.

In einem Datenbanksystem lassen sich mehrere Datenbanken anlegen, sinnvoll ist es, für jedes Projekt eine eigene Datenbank zu verwenden. Pro Datenbank können Sie dann wiederum beliebig viele Tabellen erstellen. Ein Beispiel, wann Sie Daten in verschiedene Tabellen aufteilen sollten, sehen Sie im Abschnitt 10.10.

Tipp

Eigentlich ist es empfehlenswert, pro Projekt eine eigene Datenbank zu erstellen. Oft ist bei Hostingpaketen aber nur eine beschränkte Anzahl von Datenbanken erlaubt. Dann empfiehlt es sich, zusammengehörige Tabellen eines Projekts mit dem gleichen Präfix zu versehen, etwa `pflanzen_lieferanten`, `pflanzen_produkte` und `akt_nachrichten`, `akt_user` etc. Dann wissen Sie immer, was zusammengehört. Über solche Präfixe können Sie auch gleichzeitig mehrere Instanzen eines Content Management Systems installieren, selbst wenn Ihnen nur eine Datenbank zur Verfügung steht.

MySQL – wie beispielsweise auch PostgreSQL oder Microsoft SQL Server – basieren auf der Datenbanksprache SQL (Standard Query Language). MySQL und PostgreSQL etc. haben dieselben Grundlagen und unterscheiden sich nur in Details; man spricht hier von verschiedenen SQL-Dialekten.

SQL stellt bestimmte Befehle zur Verwaltung der Datenbank zur Verfügung: zum Anlegen einer Datenbank, zur Erstellung der benötigten Tabellen und zum Ändern von Tabellen. Außerdem gibt es Befehle zum Einfügen von Daten in die Datenbank. Ganz wichtig und mächtig sind die Befehle, um bestimmte Datensätze – gefiltert nach unterschiedlichen Kriterien – auszulesen. Befehle zum Auslesen von Daten nennt man *Abfragen*.

Das Auslesen von Daten lässt sich über so genannte Indizes beschleunigen. Beim Erstellen von Tabellen können Sie einen *Index* für Spalten definieren, nach denen häufig abgefragt wird. Bei einer Adresstabelle wäre es beispielsweise sinnvoll, einen Index für das Feld mit dem Nachnamen zu erstellen. Damit sind Abfragen nach Nachnamen schneller.

Ein besonderer Index ist der *Primärschlüssel*. Dies ist ein Index, der für sich die eindeutige Identifizierbarkeit eines Datensatzes erlaubt. Häufig wird für den Primärschlüssel eine fortlaufende Nummer verwendet. Der Nachname in einer Adresstabelle wäre kein guter Kandidat für einen Primärschlüssel, da es ja mehrere Personen mit denselben Nachnamen gibt. Auch die Kombination von Vorname und Nachname garantiert nicht die eindeutige Identifizierung. Deswegen ist in vielen Fällen ein eigenes Feld mit einer Nummer am besten. Eine Kundennummer oder eine Auftragsnummer sind ebenfalls gute Kandidaten für Primärschlüssel.

Wie ist der normale Ablauf, wenn Sie mit MySQL arbeiten? MySQL besteht aus dem MySQL Server und verschiedenen MySQL-Client-Programmen. Über die MySQL-Clientprogramme senden Sie die MySQL-Befehle an den MySQL-Server und erhalten das Ergebnis zurück. Zum Umfang von MySQL gehört auch das Clientprogramm *mysql* (Achtung, Kleinschreibung!). Dies ist ein recht spartanisches Kommandozeilentool.

Ein komfortables Clientprogramm, das eine Arbeit in einer grafischen Oberfläche ermöglicht, ist *phpMyAdmin*. phpMyAdmin ist ein in PHP geschriebenes Tool zur Verwaltung von MySQL-Datenbanken. Es ist bei den meisten Providern vorinstalliert und auch bei XAMPP mit dabei.

Wenn Sie mit MySQL arbeiten, werden Sie einen Teil der Kommunikation mit dem Datenbankserver über phpMyAdmin erledigen und den anderen großen Teil über Ihre selbst geschriebenen PHP-Skripte. In Ihren PHP-Skripten müssen Sie die MySQL-Befehle als Strings abspeichern und über vorgegebene PHP-Befehle an den MySQL-Server senden.

Im Normalfall läuft das so ab: Sie erstellen eine Datenbank und die dazugehörigen Tabellen in phpMyAdmin. Um die Daten auszulesen und auf einer Webseite auszugeben, programmieren Sie ein PHP-Skript mit den Befehlen zum Auslesen der Daten.

Das heißt, die Befehle zum Auswählen, Eingeben und Löschen von Datensätzen werden Sie üblicherweise in Ihrem PHP-Skript selbst schreiben. Hingegen werden Sie das Anlegen von Datenbank und Tabellen eher über phpMyAdmin erledigen. Trotzdem sollten Sie auch den grundlegenden MySQL-Befehl zum Anlegen einer Tabelle kennen (beispielsweise weil Sie eine temporäre Tabelle in Ihrem Skript erstellen), aber Sie werden ihn nicht so häufig brauchen wie die Befehle für das Abfragen von Daten.

Deswegen werden Sie im Folgenden beides kennen lernen: die Bedienung von phpMyAdmin und die eigentlichen MySQL-Befehle.

10.3 phpMyAdmin

Bei XAMPP erreichen Sie phpMyAdmin in der Navigation unter der Oberkategorie TOOLS. Alternativ dazu geben Sie *http://localhost/phpmyadmin/* in die Adresszeile Ihres Browsers ein.

Hinweis

Bei Ihrem Provider sollte phpMyAdmin eigentlich schon vorinstalliert sein. Falls das nicht der Fall ist, finden Sie unter *http://wiki.cihar.com/pma/Quick_Install* Informationen zur Installation.

Standardmäßig, wenn Sie phpMyAdmin unter XAMPP installiert haben, läuft MySQL mit dem Benutzernamen *root* und ohne Passwort. In Kapitel 2 wurde schon darauf hingewiesen, dass XAMPP kein System für den produktiven Einsatz ist. Trotzdem sollten Sie zumindest ein Passwort für root vergeben.

Dafür ist zweierlei notwendig. Zuerst einmal müssen Sie MySQL das neue Passwort mitteilen und dann dieses Passwort in der Konfiguration von phpMyAdmin angeben.

10.3.1 root-Passwort vergeben

Unter Windows öffnen Sie die Eingabeaufforderung über START/ALLE PROGRAMME/ZUBEHÖR/EINGABEAUFFORDERUNG und wechseln dann in das Verzeichnis *xampp\mysql\bin*.

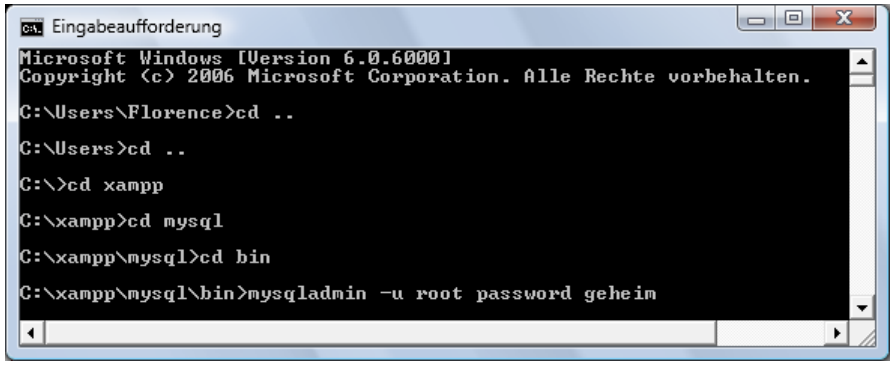


Abbildung 10.2: Das Passwort für root in MySQL setzen

Tipp

Durch `cd ..` kommen Sie ein Verzeichnis höher, `cd Verzeichnisname` wechselt in das angegebene Verzeichnis.

Geben Sie dann den Befehl `mysqladmin -u root password geheim` ein. Statt `geheim` schreiben Sie dabei Ihr Passwort.

Jetzt müssen Sie phpMyAdmin noch das neue Passwort mitteilen. Dafür editieren Sie die Konfigurationsdatei *config.inc.php*, die in *xampp\phpMyAdmin* zu finden ist. Suchen Sie hier die Stelle

```
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
```

Und ergänzen Sie Ihr Passwort:

```
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'geheim';
```

Speichern Sie die Datei. Anschließend muss der MySQL-Server gestoppt und neu gestartet werden.

Unter Linux können Sie ein Passwort für den Nutzer root komfortabel bei den allgemeinen Sicherheitschecks vergeben. Wenn Sie `/opt/lampp/lampp security` in der Kommandozeile eingeben, werden Sie Schritt für Schritt durch eine Absicherung von XAMPP geführt. Hier können Sie dann auch ein Passwort für MySQL setzen. Mehr Informationen dazu unter *http://www.apachefriends.org/de/faq-xampp-linux.html#sicherer*.

Weitergehende Sicherheit erhalten Sie, wenn Sie für die verschiedenen Nutzungen von MySQL eigene Benutzer anlegen. Bei den einzelnen Benutzern können Sie dann genau bestimmen, was diese bei einzelnen Datenbanken und einzelnen Tabellen tun dürfen oder eben nicht (siehe hierzu *http://dev.mysql.com/doc/refman/5.1/de/adding-users.html*). Bei vielen Standardhostingangeboten haben Sie dazu jedoch keine Rechte.

Nach diesen Vorbereitungen kann es jetzt losgehen. Sie erfahren alle notwendigen Schritte zum Anlegen, Füllen und Abfragen einer Datenbank.

10.4 Datenbank anlegen und benutzen

Zuerst brauchen Sie eine Datenbank. In dieser Datenbank werden Sie die einzelnen Tabellen mit den Daten erstellen. Bei Ihrem Provider kann es – je nach gewähltem Hostingpaket sein, dass Sie nur eine Datenbank verwenden dürfen und diese für Sie schon angelegt ist. Dann entfällt der Schritt, die Datenbank anzulegen.

Zum Anlegen einer Datenbank in phpMyAdmin rufen Sie die Startseite von phpMyAdmin auf. Hier finden Sie den Punkt NEUE DATENBANK ANLEGEN. Vergeben Sie dann einen Namen für die Datenbank, beispielsweise *garten*. Außerdem können Sie die KOLLATION festlegen.

Die Kollation bestimmt die Sortierung von sprachspezifischen Sonderzeichen, wie das deutsche Ü, Ä, Ö und ß. Damit deutsche Sonderzeichen korrekt im Alphabet einsortiert werden, müssen Sie hier *latin1_german1_ci* oder *latin1_german2_ci* wählen. Wenn Sie hier nichts angeben, wird die Standardkollation der MySQL-Installation genommen, die *latin1_swedish_ci* ist, also die schwedische Sortierung.

Hinweis

latin1_german1_ci behandelt von der Sortierung her *Ä* wie *A*, *Ö* wie *O*, *Ü* wie *U* und *ß* wie *s*. Bei *latin1_german2_ci* werden hingegen *Ä* wie *AE*, *Ö* wie *Oe*, *Ü* wie *Ue* und *ß* wie *ss* behandelt.

Damit bestimmen Sie auch gleichzeitig Latin1 als Zeichensatz, was im Wesentlichen ISO-8859-1 entspricht.

Wenn Sie den Namen der Datenbank – für unsere Beispielanwendung lautet er `garten` – angegeben haben und als Sortierung *latin1_german1_ci* gewählt haben, klicken Sie auf ANLEGEN.

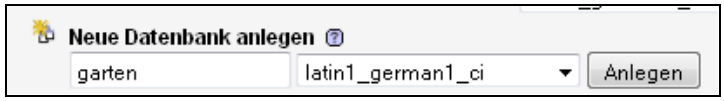


Abbildung 10.3: Datenbank in phpMyAdmin erstellen

Wenn Sie die Datenbank angelegt haben, müssen Sie auch MySQL mitteilen, dass Sie diese jetzt benutzen wollen. In phpMyAdmin geht das fast automatisch. Nachdem Sie die Datenbank angelegt haben, ist sie auch schon ausgewählt. Falls Sie eine andere Datenbank auswählen wollen, können Sie diese aus dem Ausklappmenü links unterhalb von DATENBANK bestimmen. Die Angabe in Klammern dahinter zeigt die Anzahl an Tabellen in der jeweiligen Datenbank.

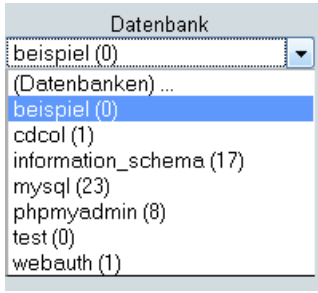


Abbildung 10.4: Liste mit vorhandenen Datenbanken

Tipp

Die Datenbanken, die Sie bereits vorfinden, sind teilweise Datenbanken, die MySQL selbst für die Verwaltung benötigt. In der Datenbank *mysql* werden beispielsweise die Benutzerrechte gespeichert.

Jetzt zu den benötigten MySQL-Befehlen. Die Datenbank `garten` erstellen Sie mit folgendem Befehl:

```
CREATE DATABASE garten;
```

Bei zusätzlicher Angabe von Zeichensatz und Kollation lautet der Befehl:

```
CREATE DATABASE garten DEFAULT CHARACTER SET latin1 COLLATE latin1_german1_ci;
```

Tipp

MySQL-Schlüsselwörter – im Listing z. B. `CREATE` und `DATABASE` – können sowohl klein- als auch großgeschrieben werden. Weil es übersichtlicher ist und gewöhnlich so gehandhabt wird, werden MySQL-Befehle hier groß geschrieben und Sie sollten sich das auch angewöhnen.

Abgeschlossen wird ein MySQL-Befehl immer durch einen Strichpunkt.

MySQL-Befehle können Sie direkt über phpMyAdmin eingeben. Klicken Sie dafür auf den Reiter SQL. Sie erhalten ein großes Textfeld, in dem Sie die Befehle eingeben und über den Button OK abschicken können.

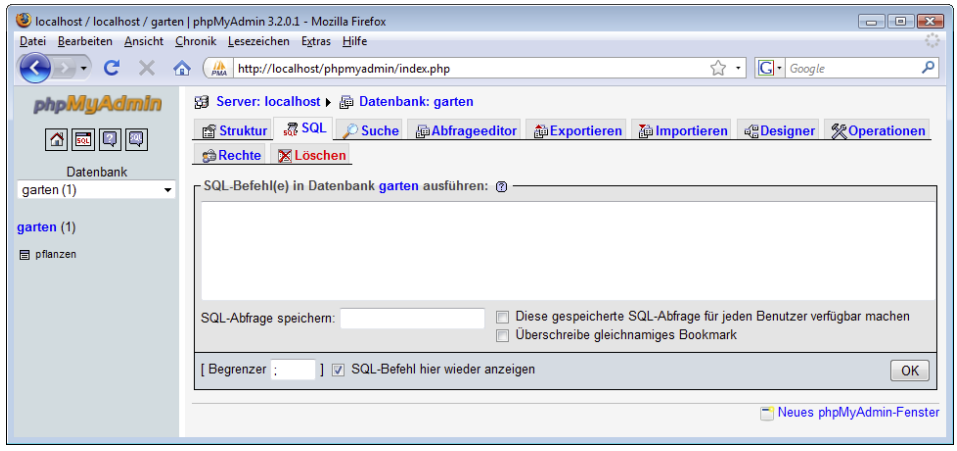


Abbildung 10.5: SQL-Befehle lassen sich über phpMyAdmin eingeben.

Um MySQL mitzuteilen, dass Sie eine bestimmte Datenbank verwenden möchten, schreiben Sie:

```
USE garten;
```

Nützlich ist ebenfalls der Befehl

```
SHOW DATABASES;
```

Damit können Sie ermitteln, welche Datenbanken bereits vorhanden sind.

10.4.1 Tabellen erstellen

Die Daten werden bei einem Datenbankmanagementsystem innerhalb von Tabellen gespeichert.

Sehen Sie zuerst, wie Sie eine Tabelle in phpMyAdmin erstellen: Kontrollieren Sie davor, dass die richtige Datenbank ausgewählt ist. Dann sehen Sie einen Bereich zum Anlegen einer Tabelle (NEUE TABELLE IN DATENBANK GARTEN ERSTELLEN). Bestimmen Sie zuerst, wie die Tabelle heißen und wie viele Spalten (Felder) sie enthalten soll. Im Beispiel heißt sie `pflanzen` und hat 4 Felder. Klicken Sie dann auf OK.

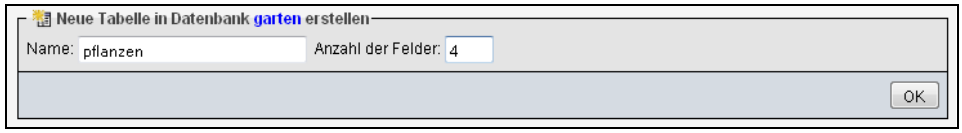


Abbildung 10.6: Neue Tabelle in phpMyAdmin anlegen

Es erscheint als Nächstes ein Fenster, in dem Sie festlegen, wie die einzelnen Felder heißen und welcher Inhalt in den einzelnen Feldern gespeichert werden soll.

Abbildung 10.7: Viele Angaben für die einzelnen Felder

- FELD bestimmt den Namen der Spalte.
- TYP ist der Inhaltstyp der Daten, also welche Daten Sie in diesem Feld speichern möchten. Genauere Informationen dazu in Abschnitt 10.5.
- LÄNGE legt bei Zeichenketten beispielsweise die mögliche Anzahl der Zeichen in einem Feld fest.
- STANDARD: Sie können einen Standardwert vergeben, der genommen wird, wenn kein anderer Wert eingegeben wird.
- KOLLATION: Sie können bei MySQL für alle Spalten einzeln festlegen, wie diese sortiert werden sollen. Ansonsten gilt die für die Datenbank insgesamt bestimmte Kollation.
- ATTRIBUTE: Bei Zahlen können Sie über `UNSIGNED` bestimmen, dass sie sich nur im positiven Bereich befinden.
- NULL: Ob die Felder auch `NULL` enthalten dürfen. Wenn in einem Feld nichts drinsteht, wird dieses dann von MySQL mit `NULL` gefüllt (und das ist unterschieden beispielsweise von der Zahl `0` und einem Leerstring).

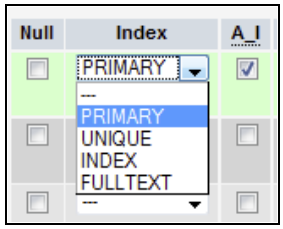


Abbildung 10.8: Optionen für Index

- Beim INDEX gibt es fünf Möglichkeiten, die sich ausschließen: PRIMARY, UNIQUE, INDEX, FULLTEXT und keins von allen vieren. Am häufigsten werden Sie mit dem Primärschlüssel zu tun haben – der ersten Option (PRIMARY). Sie können aber auch einen Index für ein Feld vergeben, nach dem häufig gesucht wird, um die Abfrage zu beschleunigen. Unique-Index ist ein Index, der eindeutig ist. FULLTEXT ist für eine Volltextsuche.
- A_I steht für `auto_increment`. Wenn Sie hier ein Häkchen machen, wird der Wert in diesem Feld automatisch bei jedem neuen Datensatz hochgezählt (inkrementiert). Das ist praktisch für das Feld mit dem Primärschlüssel zur eindeutigen Identifizierung eines Datensatzes.
- KOMMENTARE für ... Kommentare.
- Die letzten drei Optionen betreffen nur die interne Darstellung von Ergebnisseiten in phpMyAdmin. Falls Sie Bilder in Ihrer Datenbank speichern, könnten Sie beispielsweise festlegen, dass diese in phpMyAdmin als kleine Vorschaubilder angezeigt werden etc.

Für die Beispieltabelle mit dem Namen `pflanzen` werden jetzt folgende Felder benötigt.

- `pfl_id` ist eine eindeutige ID für jede Pflanze. Diese soll eine fortlaufende Nummer sein, eine ganze (große) Zahl, deswegen wird `INT` als Typ gewählt. Das Attribut `UNSIGNED` legt fest, dass nur positive Zahlen möglich sind. NULL kreuzen Sie *nicht* an, das bedeutet, dass hier immer ein Wert eingetragen wird. Bei INDEX wählen Sie PRIMARY und machen ein Häkchen bei A_I (`auto_increment`).
- `name` enthält später den Namen der Pflanze. Hier wird als Typ `VARCHAR` gewählt. Dies bedeutet, dass hier Zeichenketten eingegeben werden können. Die Länge wird auf 50 beschränkt.
- `beschreibung` enthält kurze Stichpunkte zur Beschreibung. Typ ist ebenfalls `VARCHAR`, die Zeichenzahl wird auf `150` festgelegt. Die Beschreibung ist nicht obligatorisch, das heißt in der Spalte NULL machen Sie ein Häkchen.
- `preis` nimmt auf, was das Produkt kostet. Hier wird als Typ `DECIMAL` gewählt. Bei LÄNGE wird über `9,2` bestimmt, dass der Betrag insgesamt nicht länger als 9 Zeichen (der Dezimalpunkt ist nicht mitgerechnet) sein kann und dass es zwei Stellen hinter dem Dezimalpunkt gibt. Da es sein kann, dass wir eine Pflanze aufnehmen wollen, deren Preis noch nicht feststeht, machen Sie ein Häkchen bei NULL.

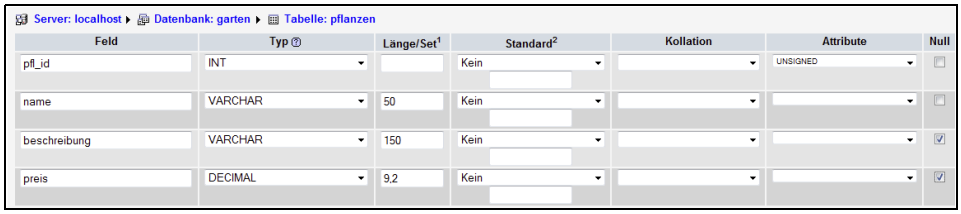


Abbildung 10.9: Die pflanzen-Tabelle mit den gewählten Eigenschaften

Wenn Sie auf SPEICHERN klicken, wird die Tabelle angelegt und phpMyAdmin zeigt Ihnen den MySQL-Befehl an, der ausgeführt wurde.

Er lautet wie in Listing 10.1 gezeigt.

Listing 10.1: Der Befehl zum Erstellen der Tabelle (tabelle_pflanzen.sql)

```
CREATE TABLE `garten`.`pflanzen` (
`pfl_id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`name` VARCHAR( 50 ) NOT NULL ,
`beschreibung` VARCHAR( 150 ) NULL ,
`preis` DECIMAL( 9, 2 ) NULL
) ENGINE = MYISAM ;
```

Sehen wir uns den Aufbau des Befehls an:

`CREATE TABLE` ist der Befehl zum Erstellen einer Tabelle. Dann folgt der Name der Tabelle. Im Beispiel wird der Name der zugehörigen Datenbank mit einem Punkt davor angegeben, was an sich nicht notwendig ist, wenn die Datenbank ausgewählt ist.

Hinweis

Der Name der Tabelle wie auch die gleich folgenden Namen der einzelnen Felder stehen innerhalb von rückwärtsgerichtete Anführungszeichen (Backticks,`). Dies ist prinzipiell nicht notwendig, solange Sie nicht eine Tabelle oder eine Spalte mit einem Namen versehen, der ein Schlüsselwort in MySQL ist. Wenn Sie beispielsweise eine Tabelle `CREATE` nennen wollen, brauchen Sie Backticks, ansonsten nicht. phpMyAdmin verwendet intern immer sicherheitshalber Backticks, sie erscheinen auch in der Ausgabe des Befehls.

In runden Klammern folgt eine Aufzählung der einzelnen Spalten: zuerst der Name, dann der Typ. Ebenfalls in runden Klammern beim Typ stehen eventuelle Längenangaben. Darauf folgt `NULL` oder `NOT NULL`. Zuletzt wird noch mit `PRIMARY KEY` der Primärschlüssel festgelegt. Außerhalb der Klammern steht hinter `ENGINE` der gewählte Typ für die Tabelle, im Beispiel `MYISAM`.

Hinweis

Es gibt mehrere unterschiedliche Tabellentypen oder Speicherengines. Sie werden am häufigsten mit MyISAM und InnoDB zu tun haben.

- MyISAM ist der Standardtyp, der außerdem Volltextsuchen unterstützt.
- InnoDB ist neuer. Dieser Typ unterstützt die Definition von Integritätsregeln bei der Arbeit mit Fremdschlüsseln (mehr dazu in Abschnitt 10.10) und Transaktionen. Mit Transaktionen können Sie sicherstellen, dass mehrere Aktionen, die eine Einheit bilden, gemeinsam erfolgreich abgeschlossen oder gesamt verworfen werden. Ein Beispiel wäre eine Buchung von einem auf ein anderes Konto: Beide Aktionen, die Abbuchung und die Gutschrift, müssen durchgeführt oder gesamt verworfen werden. Es ist hingegen nicht erwünscht, dass nur die Abbuchung erfolgt, ohne dass gleichzeitig auch die Gutschrift stattfindet.

Bevor Sie weitere mögliche Datentypen für die Felder von MySQL-Tabellen kennen lernen, ein Hinweis zur Schreibweise der MySQL-Befehle. Dass MySQL-Schlüsselwörter groß- oder kleingeschrieben werden können, haben Sie bereits erfahren. Wie sieht es aber mit der Groß-/Kleinschreibung von anderen Bezeichnern aus? Jede Tabelle entspricht einer Datei im Dateisystem, je nach Tabellentyp kann sie auch mehreren entsprechen. Deswegen gelten für die Groß-/Kleinschreibung von Datenbanken und Tabellen dieselben Regeln, die auch für das zugrunde liegende Betriebssystem gelten: Unter Windows spielt die Groß/Kleinschreibung keine Rolle, unter Linux/Unix ist sie relevant. Um kompatiblen Code zu schreiben, sollten Sie die Groß/Kleinschreibung beachten. Am einfachsten ist es, wenn Sie für Tabellen und Datenbanken kleingeschriebene Namen wählen. Bei Feldnamen, also Namen von Spalten ist die Groß/Kleinschreibung prinzipiell nicht relevant.

10.5 Datentypen in MySQL für Tabellen

In der `pflanzen`-Tabelle wurden verschiedene Datentypen eingesetzt. `INT` für den Primärschlüssel, `VARCHAR` für `name` und `beschreibung` sowie `DECIMAL` für den `preis`. Welche weiteren Datentypen Sie in MySQL verwenden können, sehen Sie jetzt im Überblick.

10.5.1 Numerische Datentypen

Es gibt verschiedene Typen für ganze Zahlen, die sich in ihrem Wertebereich unterscheiden. Standardmäßig können sie positive und negative Werte aufnehmen. Sollen nur positive Werte gespeichert werden, geben Sie `UNSIGNED` an. Eine weitere Option ist `ZEROFILL`, dann füllt MySQL leere Stellen mit `0` und automatisch wird von MySQL das Attribut `UNSIGNED` hinzugefügt.

In der folgenden Tabelle finden Sie die möglichen Datentypen für ganze Zahlen. In Klammern hinter den MySQL-Schlüsselwörtern steht `M`. `M` ist optional. Dadurch lässt sich die Anzeigenbreite definieren. Wenn ein Wert eingegeben wird, der kleiner ist als der für die Spalte festgelegte Wert, wird dieser dann mit Leerstellen bis auf die bei `M` bestimmte Breite gefüllt.

Schlüsselwort	Erläuterung
`TINYINT(M)`	Sehr kleiner Integer, der Werte zwischen -128 und 127 aufnehmen kann. Bei `UNSIGNED` Werte zwischen 0 und 255.
`SMALLINT(M)`	Kleiner Integer mit Wertebereich von -32768 bis 32767. Bei `UNSIGNED` Werte von 0 und 65535.
`MEDIUMINT(M)`	Mittelgroßer Integer mit Wertebereich von -8388608 und 8388607. Bei `UNSIGNED` Werte von 0 und 16777215.
`INT(M)`	Integer mit Wertebereich von -2147483648 und 2147483647. Bei `UNSIGNED` Werte von 0 bis 4294967295.
`INTEGER(M)`	Synonym zu `INT(M)`
`BIGINT(M)`	Großer Integer mit Wertebereich von -9223372036854775808 und 9223372036854775807. Bei `UNSIGNED` Werte von 0 bis 18446744073709551615.

Tabelle 10.1: Mögliche Datentypen für ganze Zahlen

Fließkommazahlen werden – genauso wie in PHP – auch in MySQL mit Punkt geschrieben. Die verschiedenen möglichen Typen von Fließkommazahlen listet die folgende Tabelle auf. `M` und `D` in Klammern können Sie verwenden, um optional mit `M` die *gesamte Anzeigenbreite* und mit `D` die Stellen nach dem Punkt (es wird ja ein Punkt verwendet) anzugeben.

Schlüsselwort	Erläuterung
`FLOAT(M,D)`	Kleine Fließkommazahl mit einfacher Genauigkeit. Mögliche Werte liegen zwischen -3.402823466E+38 und -1.175494351E-38, 0 und der Bereich zwischen 1.175494351E-38 und 3.402823466E+38.
`DOUBLE(M,D)`	Fließkommazahl mit doppelter Genauigkeit. Mögliche Werte liegen zwischen -1.7976931348623157E+308 und -2.2250738585072014E-308, 0 und der Bereich zwischen 2.2250738585072014E-308 und 1.7976931348623157E+308
`DECIMAL(M,D)`	Festkommazahl
`DEC(M,D)`	Synonym für `DECIMAL`
`NUMERIC(M,D)`	Synonym für `DECIMAL`
`FIXED(M,D)`	Synonym für `DECIMAL`

Tabelle 10.2: Mögliche Datentypen für Fließkommazahlen

10.5.2 Datums- und Zeittypen

Verschiedene Typen stehen zur Speicherung von Datums- und Zeitwerten zur Verfügung. `TIMESTAMP` nimmt eine Sonderrolle ein. Dieser Wert ist praktisch, um die letzte Änderung zu dokumentieren. Denn standardmäßig wird die erste `TIMESTAMP`-Zelle einer Tabelle automatisch auf Datum/Uhrzeit der letzten Änderungsaktion gesetzt. Wenn Sie einem Feld vom Typ `TIMESTAMP` als Wert `NULL` zuweisen, wird dieses auf den aktuellen Wert für Datum/Uhrzeit gesetzt.

Schlüsselwort	Erläuterung
`DATE`	Datum im Format `YYY-MM-DD`. Mögliche Werte zwischen 1000-01-01 und 9999-12-31.
`DATETIME`	Datum-Zeit im Format `YYYY-MM-DD HH:MM:SS`: Mögliche Werte zwischen '1000-01-01 00:00:00' und '9999-12-31 23:59:59'
`TIMESTAMP`	Zeitstempel. Mögliche Werte zwischen '1970-01-01 00:00:00' und einem Zeitpunkt Jahr 2037.
`TIME`	Zeitangabe. Mögliche Werte zwischen '-838:59:59' und '838:59:59'.
`YEAR`	Jahr im Format `YYYY`.

Tabelle 10.3: Mögliche Datentypen für Fließkommazahlen

In Datumsspalten können Sie die Zeit/Datumsangabe wahlweise als Zahl oder als String angeben.

10.5.3 Datentypen für Strings

Auch für Strings gibt es unterschiedliche Datentypen, die Sie in der folgenden Tabelle aufgelistet finden. `M` bestimmt die maximale Spaltenlänge. `CHAR` ist ein String fester Länge. Bei der Speicherung wird eine Zeichenkette, die weniger Zeichen enthält, als bei `M` angegeben, durch Leerzeichen auf die Länge `M` gefüllt. Diese Auffüllung geschieht nicht beim Typ `VARCHAR(M)`.

Schlüsselwort	Erläuterung
`CHAR(M)`	String fester Länge
`VARCHAR(M)`	String variabler Länge
`TINYTEXT`	Maximallänge: 255 Zeichen
`TEXT`	Maximallänge: 65.535 Zeichen
`MEDIUMTEXT`	Maximallänge von 16.777.215 Zeichen
`LONGTEXT`	Maximallänge von 4.294.967.295 Zeichen

Tabelle 10.4: Mögliche Datentypen für Strings

Eine Sonderrolle nehmen ENUM und SET ein.

Schlüsselwort	Erläuterung
ENUM('wert1', 'wert2', ...)	Auflistung möglicher Werte, von denen in einem konkreten Feld immer nur einer gespeichert wird.
SET('wert', 'wert2', ...)	Menge möglicher Werte. In einem konkreten Feld können mehrerer dieser Werte aufgeführt werden.

Tabelle 10.5: Auflistung und Menge an möglichen Werten

10.5.4 Binärdaten

Auch zum Speichern von Binärdaten wie beispielsweise für Bilder gibt es mehrere Datentypen.

Schlüsselwort	Erläuterung
TINYBLOB	Binärdaten mit Maximallänge von 255 Byte
BLOB	Binärdaten mit Maximallänge von 65.535 Byte
MEDIUMBLOB	Binärdaten mit Maximallänge von 16.777.215 Byte
LONGBLOB	Binärdaten mit Maximallänge von 4 GByte

Tabelle 10.6: Mögliche Datentypen für binäre Daten

Tipp

Häufig speichert man aber – beispielsweise bei einer Bildergalerie, die man über eine Datenbank verwaltet – in der Datenbank nicht die Bilder selbst, sondern nur ihren Pfad.

10.6 Daten einfügen

Wenn die Tabellen erstellt sind, müssen sie mit Daten gefüllt werden. Diese können aus verschiedenen Quellen stammen:

- Handelt es sich um eine überschaubare Anzahl an Datensätzen, können Sie diese händisch in phpMyAdmin eingeben.
- Daten können auch aus Formularen stammen, die die Benutzer ausfüllen. Hierfür benötigen Sie den MySQL-INSERT-Befehl, den Sie gleich kennen lernen.

- Wenn Sie die Daten bereits in einer Exceltabelle o. Ä. vorliegen haben, können Sie diese über phpMyAdmin importieren. Hierfür muss die Datei als CSV (comma-separated-value) abgespeichert werden. Diese Option bieten alle gängigen Tabellenkalkulationsprogramme. Die Importoptionen finden Sie in phpMyAdmin unter IMPORTIEREN.

Sehen wir uns zuerst an, wie Sie Daten in phpMyAdmin eintragen. Dafür wählen Sie die Tabelle aus, in die Sie Daten eintragen möchten und klicken in der horizontalen Navigation auf EINFÜGEN. Sie erhalten ein Formular zum Eintragen der Daten. Die Inhalte fügen Sie unterhalb von WERT ein. Da MySQL den Wert für die `pfl_id` selbst vergeben soll, lassen Sie dieses Feld leer.

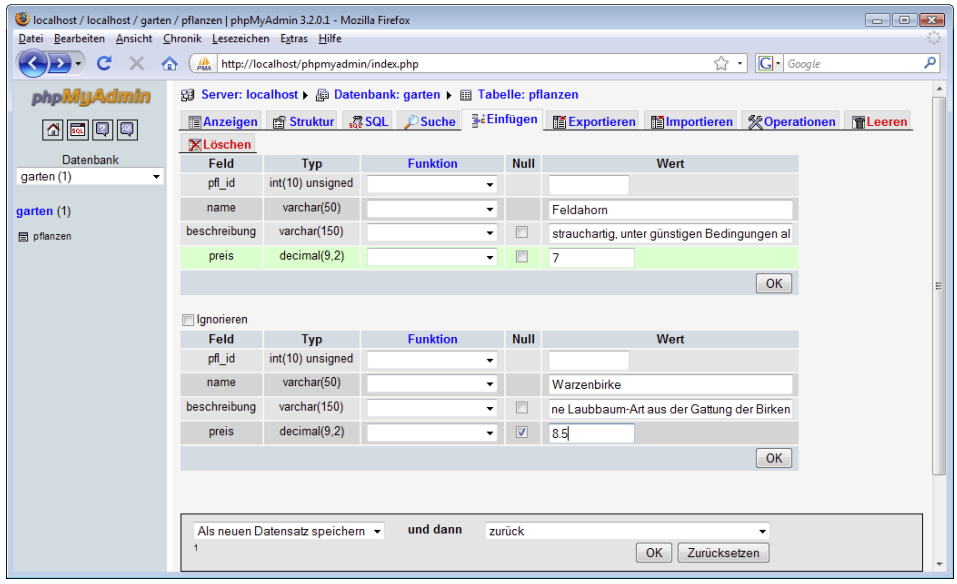


Abbildung 10.10: Daten eingeben über phpMyAdmin

Standardmäßig werden zwei Formulare zum Eintragen von zwei Datensätzen angezeigt. Sie können auch nur das obere ausfüllen und dann auf OK drücken. Wollen Sie gleich weitere eingeben, wählen Sie unten anstelle von ZURÜCK die Option ANSCHLIEßEND EINEN WEITEREN DATENSATZ EINFÜGEN.

Dann sehen Sie wieder den MySQL-Befehl. Wenn man die von phpMyAdmin immer um Tabellennamen und Spalten eingefügten Backticks entfernt, die hier nicht notwendig sind, sieht er folgendermaßen aus:

Listing 10.2: Datensätze eintragen (tabelle_pflanzen_zweieintragen.sql)

```
INSERT INTO pflanzen (
pfl_id,
name,
beschreibung,
preis
)
VALUES (
NULL , 'Feldahorn', 'strauchartig, unter günstigen Bedingungen als Baum mit Höhen
zwischen 10 und 15 Metern', '7'
), (
NULL , 'Warzenbirke', 'sommergrüne Laubbaum-Art aus der Gattung der Birken', '8.5'
);
```

So lautet der Befehl für das gleichzeitige Eintragen von zwei Datensätzen: Hinter `INSERT INTO` folgt der Name der Tabelle. In runden Klammern listen Sie durch Komma getrennt die Spaltennamen auf, in die Sie Daten eintragen möchten. Dann folgt das Schlüsselwort `VALUES` und in Klammern dahinter die Daten, die eingetragen werden sollen. Diese müssen der vorher bei den Spalten angegebenen Reihenfolge entsprechen. Für das Feld `pfl_id` geben Sie `NULL` als Wert an, damit MySQL den Autoinkrement-Wert selbst vergibt. Im Beispiel sehen Sie, dass alle Werte in Hochkommata stehen. Das ist bei Strings obligatorisch, bei Zahlen wie dem Preis im Beispiel benötigen Sie das Hochkomma nicht, können es aber trotzdem einsetzen.

Soll nur ein Datensatz eintragen werden, geben Sie hinter `VALUES` keine zweite runde Klammer mit Werten an.

```
INSERT INTO pflanzen (
pfl_id,
name,
beschreibung,
preis
)
VALUES (
NULL , 'Feldahorn', 'strauchartig, unter günstigen Bedingungen als Baum mit Höhen
zwischen 10 und 15 Metern', '7'
);
```

Wenn Sie Werte in allen Spalten und in genau der Reihenfolge eintragen möchten, in der diese auch in der Tabelle stehen, können Sie die Spaltennamen auch weglassen:

```
INSERT INTO pflanzen
VALUES (
NULL, 'Feldahorn', 'strauchartig, unter günstigen Bedingungen als Baum mit Höhen
zwischen 10 und 15 Metern', '7'
)
```

Tipp

Die Schreibweise mit der Angabe der Spaltennamen hat jedoch den Vorteil, dass der `INSERT`-Vorgang auch funktioniert, falls die Reihenfolge der Spalten in der Tabelle verändert wird oder neue Spalten hinzukommen.

10.7 Datensätze verändern

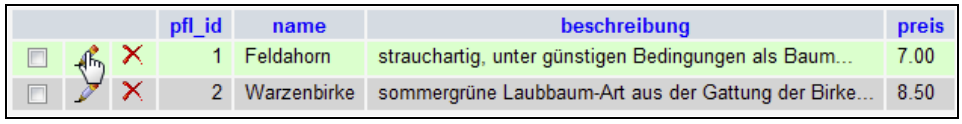
Bestehende Datensätze lassen sich auch ändern. Lassen Sie sich hierfür zuerst den Inhalt der Tabelle anzeigen. Wenn Sie Ihre Datenbank ausgewählt haben, sehen Sie die angelegten Tabellen und können neben dem Tabellennamen auf das Symbol für ANZEIGEN klicken.

Abbildung 10.11: Das Symbol neben dem Namen der Tabelle steht für Anzeigen.

Sind Sie bereits bei der Bearbeitung Ihrer Tabelle, klicken Sie im horizontalen Menü auf ANZEIGEN.

Abbildung 10.12: Daten anzeigen lassen im Menü der Tabelle

Sie sehen nun die bisher eingegebenen Dateien. Ein Klick auf das Stiftsymbol vor einem Datensatz öffnet ein neues Formular, in dem Sie die bestehenden Inhalte modifizieren können.



	pfl_id	name	beschreibung	preis
	1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum...	7.00
	2	Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birke...	8.50

Abbildung 10.13: Stiftsymbol zum Bearbeiten eines Datensatzes

Der zugehörige MySQL-Befehl lautet:

```
UPDATE pflanzen
SET preis = '5.50'
WHERE pfl_id =1 LIMIT 1 ;
```

Hinter dem MySQL-Schlüsselwort `UPDATE` steht der Name der Tabelle und dann folgt `SET` mit Spaltennamen und neuem Wert. Mit der `WHERE`-Klausel schränken Sie ein, welchen Datensatz oder Datensätze Sie verändern wollen, im Beispiel ist es nur der Datensatz mit `pfl_id = 1`. `LIMIT 1` schließlich begrenzt die Änderung zusätzlich auf genau einen Datensatz. Mehr zur `WHERE`-Klausel lesen Sie in Abschnitt 10.9.2.

Achtung

Verwenden Sie `UPDATE` ohne `WHERE`-Klausel, so führen Sie die Änderung bei allen Datensätzen der Tabelle durch.

10.8 Datensätze löschen

Neben dem Symbol zum Bearbeiten eines Datensatzes finden Sie auch das rote Kreuz, um einen Datensatz zu löschen. Bevor der Datensatz gelöscht wird, fragt phpMyAdmin noch einmal nach, ob Sie das wirklich wollen.

Der MySQL-Befehl lautet beispielsweise:

```
DELETE FROM pflanzen
WHERE pfl_id = 3 LIMIT 1;
```

Damit löschen Sie den entsprechenden Datensatz. Wieder ist es wichtig, eine `WHERE`-Klausel anzugeben, um den Löschvorgang auf bestimmte Datensätze zu beschränken.

Tipp

Wenn Sie einen Datensatz löschen, erhalten Sie eine nicht mehr durchgehende Folge von Auto-Inkrementwerten. Das braucht Sie aber nicht zu irritieren – denn das ist das gewünschte Verhalten. Ein Primärschlüssel dient der eindeutigen Kennzeichnung eines Datensatzes und darf sich nicht ändern, wenn ein anderer Datensatz gelöscht wird.

10.9 Daten auslesen

Es gibt viele Möglichkeiten, die Daten auslesen zu lassen. Sie können die Daten nach unterschiedlichen Kriterien filtern, sie sortieren und auch Daten von mehreren Tabel-

len auslesen. Das Auslesen der Daten werden Sie im Normalfall in einem PHP-Skript vornehmen, das heißt, hierfür müssen Sie die zugrunde liegenden MySQL-Befehle kennen.

Tipp

Um die Beispiele dieses Unterkapitels nachvollziehen zu können, brauchen Sie eine Tabelle mit etwas mehr Daten. Sie finden bei den Listings dieses Buchs auch die Datei *tabelle_pflanzen_mit_daten.sql*, die den vollständigen MySQL-Code zur Erstellung der Tabelle und zum Einfügen von acht Datensätzen enthält. Am einfachsten ist es, Sie löschen Ihre bisherige Tabelle und erstellen dann die Tabelle mit den Daten neu.

Zum Löschen der Tabelle in phpMyAdmin klicken Sie in phpMyAdmin im linken Bereich Ihre Datenbank an, so dass rechts die Tabellen angezeigt werden. Dann können Sie durch einen Klick auf das rote Kreuz die Tabelle löschen.

Abbildung 10.14: Tabelle löschen über phpMyAdmin

Der zugehörige Befehl lautet:

```
DROP TABLE pflanzen;
```

Wechseln Sie dann zum SQL-Eingabefenster (Menüpunkt SQL in der horizontalen Menüleiste), fügen Sie den aus dem Listing *tabelle_pflanzen_mit_daten.sql* kopierten Code ein und bestätigen Sie mit OK.

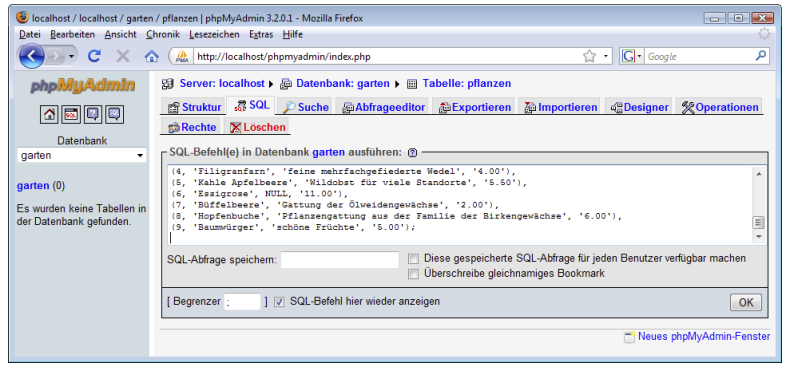


Abbildung 10.15: Die kopierten Befehle zum Erstellen und Befüllen der Tabelle können bei SQL in das Feld eingefügt und ausgeführt werden.

Jetzt zu den Befehlen zum Auslesen von Daten. Verwenden Sie zum Testen der Befehle wieder den Menüpunkt `SQL`. Um alle Daten der Tabelle `pflanzen` auszulesen, schreiben Sie:

```
SELECT * FROM pflanzen;
```

Ein Klick auf OK zeigt das Ergebnis an.

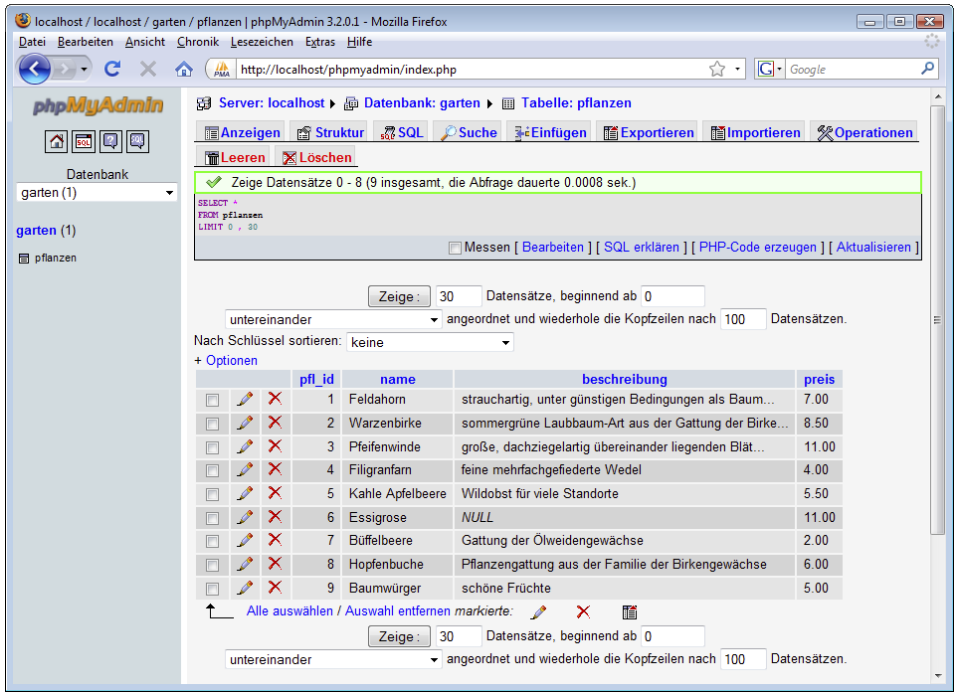


Abbildung 10.16: Alle Datensätze ausgeben lassen

Wenn Sie hingegen nicht alle Spalten, sondern nur ausgewählte ausgeben lassen möchten, so notieren Sie diese anstelle des Sternchens:

```
SELECT name, preis FROM pflanzen;
```

Tipp

Außer Sie benötigen wirklich alle Spalten, sollten Sie `SELECT *` meiden, weil es ineffektiv ist.

10.9.1 Datensätze sortieren und Anzahl beschränken

Mit `ORDER BY` bestimmen Sie die Sortierung:

```
SELECT name, preis FROM pflanzen ORDER BY preis;
```

	name	preis
	Büffelbeere	2.00
	Filigranfarn	4.00
	Baumwürger	5.00
	Kahle Apfelbeere	5.50
	Hopfenbuche	6.00
	Feldahorn	7.00
	Warzenbirke	8.50
	Pfeifenwinde	11.00
	Essigrose	11.00

Abbildung 10.17: Die Pflanzen sind aufsteigend nach Preisen sortiert.

Abbildung 10.17 zeigt, dass Pflanzen jetzt nach Preisen sortiert sind. Soll bei gleichen Preisen noch ein weiteres Sortierkriterium angegeben werden, so schreiben Sie dieses nach einem Komma.

```
SELECT name, preis FROM pflanzen ORDER BY preis, name;
```

Das wirkt sich nur bei Pfeifenwinde und Essigrose aus, die beide denselben Preis haben: Dann wird die *Essigrose* vor der *Pfeifenwinde* angezeigt.

Um die Sortierreihenfolge umzukehren, schreiben Sie `DESC`.

```
SELECT name, preis FROM pflanzen ORDER BY preis DESC;
```

`LIMIT` begrenzt die Anzahl der angezeigten Datensätze. Mit dem folgenden Befehl werden die drei teuersten Pflanzen ausgegeben.

```
SELECT name, preis FROM pflanzen ORDER BY preis DESC LIMIT 3;
```

Wenn Sie hinter `LIMIT` *einen Wert* angeben, begrenzt dieser die Anzahl der zurückgegebenen Datensätze. Sie können hier auch *zwei* durch Komma getrennte Werte schreiben: Dann gibt der erste an, ab welcher Stelle die Daten zurückgegeben werden sollen, und der zweite, wie viele.

Mit `SELECT DISTINCT` entfernen Sie Duplikate. Wenn beispielsweise in einer Tabelle `lieferanten` die Adressen von den Lieferanten gespeichert sind, können Sie sich durch folgende Abfrage eine Liste der Orte ausgeben lassen, in denen es Lieferanten gibt. Durch `DISTINCT` werden Mehrfachnennungen unterdrückt.

```
SELECT DISTINCT ort
FROM lieferanten
ORDER BY ort;
```

10.9.2 Datensätze auswählen und filtern

Jetzt können Sie die Daten nach verschiedenen Kriterien filtern. Die Bedingungen werden hinter `WHERE` notiert. Durch folgenden Befehl erhalten Sie z. B. nur Pflanzen, deren Preis größer als 9 ist.

```
SELECT name, preis FROM pflanzen WHERE preis > 9;
```

Mit = überprüfen Sie auf Übereinstimmung:

```
SELECT * FROM pflanzen WHERE name = "Büffelbeere";
```

Bei der Übereinstimmung ist Groß-/Kleinschreibung nicht relevant. Soll etwas nicht übereinstimmen, verwenden Sie <>.

Eine Besonderheit gibt es bei der Überprüfung, ob in einer Spalte *nichts* steht. Um das nachzuvollziehen, brauchen wir einen Datensatz, bei dem in einem Feld nichts steht. Das ist bereits bei der Essigrose der Fall, hier gibt es keine Beschreibung.

Um den Datensatz ohne Beschreibung zu finden, benötigen Sie folgenden Befehl:

```
SELECT *
FROM pflanzen
WHERE beschreibung IS NULL;
```

Entsprechend erhalten Sie mit `IS NOT NULL` alle Datensätze, in denen etwas bei der Beschreibung steht:

```
SELECT *
FROM pflanzen
WHERE beschreibung IS NOT NULL;
```

Achtung

Eine Überprüfung mit `beschreibung = NULL` oder `beschreibung <> NULL` führt hingegen nicht zum gewünschten Ergebnis!

Eben haben Sie gesehen, wie Sie auf genaue Übereinstimmung prüfen können. Sie können bei Vergleichen jedoch auch Platzhalter einsetzen. Der Unterstrich _ steht

dabei für *ein* beliebiges Zeichen, das Prozentzeichen `%` für *beliebig viele* beliebige Zeichen. Beim Einsatz der Platzhalter benötigen Sie die Schlüsselwörter `LIKE` und `NOT LIKE` zum Vergleich.

Mit der folgenden Abfrage erhalten Sie alle Pflanzen, deren Name `beere` enthält.

```
SELECT *
FROM pflanzen
WHERE name LIKE '%beere%';
```

Tipp

Sie können in MySQL nach Übereinstimmungen auch mithilfe von regulären Ausdrücken suchen. Mehr hierzu im MySQL-Manual unter *http://dev.mysql.com/doc/refman/5.1/de/pattern-matching.html*.

Hinweis

Prinzipiell ist die Suche mit `LIKE` aber ineffektiv – besser ist eine Volltextsuche. Mehr Informationen zur Volltextsuche, die nur im MyISAM- und nicht beim InnoDB-Tabellentyp funktioniert, finden Sie unter *http://dev.mysql.com/doc/refman/5.1/de/fulltext-search.html*.

Mehrere Bedingungen können durch `AND` oder `OR` verknüpft werden. Über folgenden Befehl erhält man alle Pflanzen, deren Name `beere` beinhaltet **oder** `ahorn`:

```
SELECT *
FROM pflanzen
WHERE name LIKE '%beere%'
OR name LIKE '%ahorn%';
```

	pfl_id	name	beschreibung	preis
	1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum...	7.00
	5	Kahle Apfelbeere	Wildobst für viele Standorte	5.50
	7	Büffelbeere	Gattung der Ölweidengewächse	2.00

Abbildung 10.18: Alle Pflanzen mit beere oder ahorn im Namen

Bei `AND` müssen beide Bedingungen zutreffen. Im folgenden Beispiel muss der Name `beere` beinhalten *und* der Preis kleiner als 4 sein.

```
SELECT *
FROM pflanzen
WHERE name LIKE '%beere%'
AND preis < 4;
```

	pfl_id	name	beschreibung	preis
	7	Büffelbeere	Gattung der Ölweidengewächse	2.00

Abbildung 10.19: Nur Pflanzen mit beere im Namen, die weniger als 4 kosten

Um Vergleiche mit mehreren Werten durchzuführen, können Sie auch `IN` benutzen. Anstelle von

```
SELECT *
FROM pflanzen
WHERE pfl_id = 1 OR pfl_id = 2 OR pfl_id = 3;
```

schreiben Sie kürzer Folgendes:

```
SELECT *
FROM pflanzen
WHERE pfl_id IN (1, 2, 3);
```

	pfl_id	name	beschreibung	preis
	1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum...	7.00
	2	Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birke...	8.50
	3	Pfeifenwinde	große, dachziegelartig übereinander liegenden Blät...	11.00

Abbildung 10.20: Die Pflanzen mit `pfl_id IN (1, 2, 3)`

Bei mehreren Bedingungen können Sie mit Klammern die Reihenfolge der Abarbeitung festlegen.

10.9.3 Datensätze zählen

Um zu ermitteln, wie viele Datensätze es sind, benutzen Sie `COUNT(*)`:

```
SELECT COUNT(*)
FROM pflanzen;
```

COUNT(*)
9

Abbildung 10.21: Das Ergebnis der Zählung mit COUNT(*)

Die Spaltenüberschrift können Sie durch `AS` und einen anderen Namen ändern:

```
SELECT COUNT(*) AS gesamt
FROM pflanzen;
```

gesamt
9

Abbildung 10.22: So ist's lesbarer – hier wurde eine andere Spaltenüberschrift mit AS angegeben.

`AS` zur Angabe eines anderen Spaltennamens funktioniert selbstverständlich nicht nur bei `COUNT(*)`, sondern bei allen Spaltennamen und Sie können es auch benutzen, um einen Alias bei einer Tabelle anzugeben.

Hinweis

Es gibt viele nützliche Funktionen, die Ihnen MySQL zur Verfügung stellt. Zum Beispiel können Sie schnell ermitteln, was der höchste, der niedrigste und der durchschnittliche Preis bei den Pflanzen ist:

```
SELECT MIN(preis) AS MinPreis, MAX(preis) AS MaxPreis,
AVG(preis) AS MittelPreis
FROM pflanzen;
```

MinPreis	MaxPreis	MittelPreis
2.00	11.00	6.666667

Abbildung 10.23: Preisstruktur der pflanzen-Tabelle

Diese Funktionen werden normalerweise in Kombination mit `GROUP BY`-Klauseln benutzt (siehe Abschnitt 10.10.1). Wenn `GROUP BY` nicht da steht wie im Beispiel hier, werden alle Datensätze gruppiert.

Eine Übersicht über die MySQL-Funktionen finden Sie im MySQL-Manual unter *http://dev.mysql.com/doc/refman/5.1/de/functions.html*.

Tipp

`SELECT` dient nicht nur dazu, Daten aus einer Tabelle auszulesen. Sie können es auch benutzen, um sich bestimmte Informationen anzeigen zu lassen oder beispielsweise Berechnungen durchzuführen.

So geben Sie die Version von MySQL aus:

```
SELECT VERSION();
```

So das aktuelle Datum inklusive Uhrzeit:

```
SELECT NOW();
```

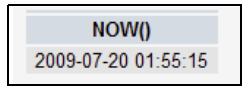


Abbildung 10.24: Beispielergebnis für SELECT NOW() in phpMyAdmin

Sie können aber auch Berechnungen durchführen und MySQL als Taschenrechner nutzen:

```
SELECT 14 * 15;
```

gibt 210 aus.

10.10 Mit mehreren Tabellen arbeiten

Nehmen wir einmal an, wir möchten unsere Informationen zu den Pflanzen noch durch die Angabe der jeweiligen Lieferanten ergänzen. Von den Lieferanten selbst benötigen wir den Lieferantennamen, die Straße, PLZ und Ort.

In einem ersten Ansatz könnte man diese Information wie folgt in der Tabelle unterbringen.

pfl_id	name	beschreibung	preis	firma	strasse	plz	ort
1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum mit Höhen zwischen 10 und 15 Metern	7	Gärtnerbedarf Müller	Dorfstraße 8	12345	Dorfen
2	Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birken	8,5	Grünes Allerlei	Stadtstraße 9	54321	Stadt
3	Pfeifenwinde	große, dachziegelartig übereinanderliegende Blätter	11	Grünes Allerlei	Stadtstraße 9	54321	Stadt
4	Filigranfarn	feine mehrfachgefiederte Wedel	4	Grüner Finger	Am Waldrand	22333	Wald
5	Kahle Apfelbeere	Wildobst für viele Standorte	5	Grüner Finger	Am Waldrand	22333	Wald

Tabelle 10.7: Pflanzentabelle mit (redundanten) Daten der Lieferanten

pfl_id	name	beschreibung	preis	firma	strasse	plz	ort
6	Essigrose	kultivierte, robuste Rosenart	11	Gärtnerbedarf Müller	Dorfstraße 8	12345	Dorfen
7	Büffelbeere	Gattung der Ölweidengewächse	2	Gärtnerbedarf Müller	Dorfstraße 8	12345	Dorfen
8	Hopfenbuche	Pflanzengattung aus der Familie der Birkengewächse	6	Grüner Finger	Am Waldrand	22333	Wald
9	Baumwürger	schöne Früchte	5	Grüner Finger	Am Waldrand	22333	Wald

Tabelle 10.7: Pflanzentabelle mit (redundanten) Daten der Lieferanten (Forts.)

Dies ist jedoch unpraktisch, weil die Informationen zu den Lieferanten *redundant* abgespeichert werden. Es kann passieren, dass die Adressen nicht konsistent eingegeben werden. Besonders deutlich zeigt sich der Nachteil dieser Lösung, wenn sich beispielsweise die Adresse von der Firma Grüner Finger ändert: Dann muss die Adresse in mehreren Zeilen angepasst werden – das ist fehleranfällig und kann zu Inkonsistenzen führen. Außerdem benötigt diese Art der Speicherung mehr Speicherplatz, das Durchsuchen und Analysieren der Daten ist mühsamer.

Besser ist es, die Informationen zu den Lieferanten in einer eigenen Tabelle unterzubringen und die Lieferanten mit einer ID zu versehen. Dann genügt es, die ID des jeweiligen Lieferanten bei jedem Datensatz der `pflanzen`-Tabelle unterzubringen.

Hinweis

Die richtige Darstellung von Daten und Aufteilung auf Tabellen wird über so genannte *Normalformen* beschrieben, den Vorgang selbst nennt man *Normalisierung*. Mehr Informationen dazu unter *http://de.wikipedia.org/wiki/Normalisierung_(Datenbank)*.

Das heißt, Sie brauchen eine weitere Tabelle mit dem Namen `lieferanten` mit fünf Feldern.

Folgende Angaben werden für die Felder gewählt:

- `liefer_id` ist vom Typ `INT`, Attribut UNSIGNED, Index `Primary` und soll automatisch hochgezählt werden (A_I).
- `firma` ist vom Typ `VARCHAR` und hat eine Länge von 50.

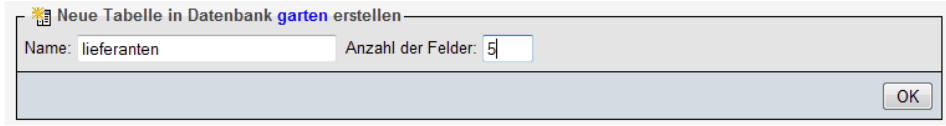


Abbildung 10.25: Die neue Tabelle lieferanten hat fünf Felder.

- `strasse` ist vom Typ `VARCHAR` mit einer Länge von 50, NULL ist erlaubt.
- `plz` ist ebenfalls vom Typ `VARCHAR`, aber mit einer Länge von 5, NULL ist erlaubt.
- `ort` ist auch vom Typ `VARCHAR` mit einer Länge von 50, NULL ist erlaubt.

Wahlweise können Sie auch den folgenden MySQL-Befehl zum Erstellen der Tabelle eingeben:

Listing 10.3: Der SQL-Code zum Erstellen der Lieferantentabelle (tabelle_lieferantenstruktur.sql)

```
CREATE TABLE lieferanten (
liefer_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
firma VARCHAR( 50 ) NOT NULL ,
strasse VARCHAR( 50 ) NULL ,
plz VARCHAR( 5 ) NULL ,
ort VARCHAR( 50 ) NULL
) ENGINE = MYISAM ;
```

Sie sehen, für die `liefer_id` wird `INT` als Typ gewählt, für die anderen Felder `VARCHAR`. Auch für `plz`, die Postleitzahl, ist `VARCHAR` der richtige Typ, denn mit Postleitzahlen führt man keine Berechnungen durch und außerdem gibt es Postleitzahlen, die mit 0 beginnen. Zwar könnte man auch bei Zahlen festlegen, dass sie mit 0 aufgefüllt werden, aber einfacher ist es, `VARCHAR` als Typ zu wählen.

Dann müssen noch die folgenden Daten eingetragen werden. Dies können Sie wieder über das Eingabeformular über phpMyAdmin machen, das Sie über EINFÜGEN erreichen. Oder Sie geben den folgenden MySQL-Befehl ein:

Listing 10.4: Die Daten für die Lieferantentabelle (tabelle_lieferantenstruktur.sql)

```
INSERT INTO lieferanten (firma, strasse, plz, ort)
VALUES ('Gärtnerbedarf Müller', 'Dorfstraße 8', '12345', 'Dorfen'),
('Grünes Allerlei', 'Stadtstraße 9', '54321', 'Stadt'),
('Grüner Finger', 'Am Waldrand', '22333', 'Wald');
```

Sie sehen im Beispiel, dass nicht alle Felder aufgezählt wurden – das Feld `liefer_id`, das ja über einen Autoinkrement-Wert von MySQL gefüllt wird, wird nicht aufgeführt. Die andere Möglichkeit, es aufzuführen, aber dann `NULL` als Wert zu bestimmen, haben Sie in Abschnitt 10.6 gesehen.

	liefer_id	firma	strasse	plz	ort
	1	Gärtnerbedarf Müller	Dorfstraße 8	12345	Dorfen
	2	Grünes Allerlei	Stadtstraße 9	54321	Stadt
	3	Grüner Finger	Am Waldrand	22333	Wald

Abbildung 10.26: Die Tabelle mit den Lieferanten

Jetzt stehen beide Tabellen, aber es fehlt noch die Verknüpfung zwischen diesen. Hierfür benötigen wir eine weitere Spalte in der Tabelle `pflanzen`, in die wir die ID des Lieferanten eintragen, der die jeweilige Pflanze liefert.

Um die Tabelle per phpMyAdmin zu ändern, wählen Sie sie erst einmal im linken Bereich aus. Die STRUKTUR der Tabelle wird angezeigt.

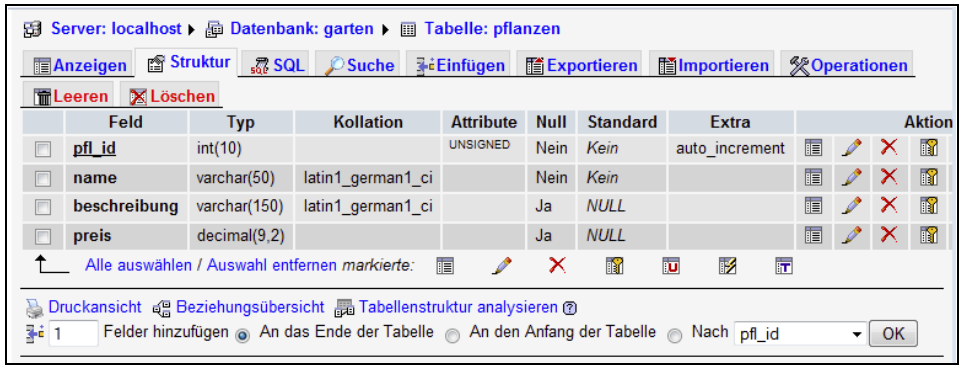


Abbildung 10.27: Die Struktur der Pflanzentabelle

Unter der Tabelle mit der Struktur erscheint 1 FELDER HINZUFÜGEN. Außerdem lässt sich noch bestimmen, an welcher Stelle das neue Feld erscheinen soll. Die Voreinstellung AN DAS ENDE DER TABELLE ist im Beispiel eine gute Wahl. Klicken Sie dann auf OK.

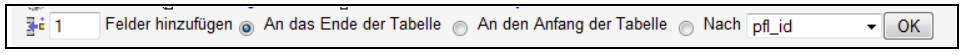


Abbildung 10.28: Über phpMyAdmin können Sie ein weiteres Feld in der Tabelle pflanzen ergänzen.

Nennen Sie das neue Feld `liefer_id`. Es ist vom Typ `INT`, `UNSIGNED` und als Default können Sie `NULL` vorgeben. Über SPEICHERN wird die Änderung der Tabelle `pflanzen` ausgeführt. Wieder sehen Sie den zugehörigen MySQL-Befehl:

Listing 10.5: Der Befehl zum Einfügen eins Felds (tabelle_pflanzen_aendern.sql)

```
ALTER TABLE `pflanzen` ADD `liefer_id` INT UNSIGNED NULL ;
```

Zum Ändern einer Tabelle dient der Befehl `ALTER TABLE`, dann folgt der Tabellenname. Über `ADD` wird ein Feld hinzugefügt.

Hinweis

Weitere Optionen für `ALTER TABLE` finden Sie wieder im MySQL-Manual *http://dev.mysql.com/doc/refman/5.1/de/alter-table.html*.

Wenn die neue Spalte in der Tabelle `pflanzen` ergänzt ist, können Sie sie mit den entsprechenden IDs der Lieferanten-Tabelle füllen.

Lassen Sie sich zuerst die Inhalte von `pflanzen` ANZEIGEN über den gleichnamigen Menüpunkt. Jetzt können Sie die einzelnen Datensätze auswählen und über das Stift-Symbol bearbeiten.

Der zugehörige MySQL-Befehl zur Eintragung der `liefer_id` bei einem Datensatz wird wieder angezeigt und lautet:

```
UPDATE `garten`.`pflanzen` SET `liefer_id` = '1' WHERE `pflanzen`.`pfl_id` =1 LIMIT 1 ;
```

Ohne Backticks lässt er sich auch so schreiben:

```
UPDATE pflanzen SET liefer_id = '1' WHERE pflanzen.pfl_id =1 LIMIT 1;
```

Tipp

Um alle Datensätze auf einmal zu bearbeiten, wählen Sie alle Datensätze über ALLE AUSWÄHLEN aus und klicken Sie dann auf das Stift-Symbol hinter MARKIERTE.

Abbildung 10.29: Über phpMyAdmin lassen sich schnell auch mehrere oder sogar alle Datensätze auf einmal bearbeiten.

Tipp

Den vollständigen Code zum Eintragen der liefer_id in der Pflanzentabelle finden Sie auch als Listing unter dem Namen *tabelle_pflanzen_lieferid_eintragen.sql*.

	pfl_id	name	beschreibung	preis	liefer_id
	1	Feldahorn	strauchartig, unter günstigen Bedingungen als Baum...	7.00	1
	2	Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birke...	8.50	2
	3	Pfeifenwinde	große, dachziegelartig übereinander liegenden Blät...	11.00	2
	4	Filigranfarn	feine mehrfachgefiederte Wedel	4.00	3
	5	Kahle Apfelbeere	Wildobst für viele Standorte	5.50	3
	6	Essigrose	*NULL*	11.00	1
	7	Büffelbeere	Gattung der Ölweidengewächse	2.00	1
	8	Hopfenbuche	Pflanzengattung aus der Familie der Birkengewächse	6.00	3
	9	Baumwürger	schöne Früchte	5.00	3

Abbildung 10.30: Die IDs der Lieferanten sind den einzelnen Pflanzen zugeordnet.

Um jetzt zu jeder Pflanze den Namen der Firma des Lieferanten anzeigen zu lassen, muss man beide Tabellen verknüpfen.

Bei `FROM` können Sie beide Tabellen angeben und außerdem müssen Sie bei `WHERE` spezifizieren, über welches Feld die beiden Tabellen verknüpft werden sollen.

```
SELECT name, firma
FROM pflanzen, lieferanten
WHERE pflanzen.liefer_id = lieferanten.liefer_id;
```

Da die Spalte `liefer_id` in beiden Tabellen existiert, müssen Sie in der `WHERE`-Klausel zusätzlich vor dem Spaltennamen mit einem Punkt den Namen der Tabelle angeben.

name	firma
Feldahorn	Gärtnerbedarf Müller
Warzenbirke	Grünes Allerlei
Pfeifenwinde	Grünes Allerlei
Filigranfarn	Grüner Finger
Kahle Apfelbeere	Grüner Finger
Essigrose	Gärtnerbedarf Müller
Büffelbeere	Gärtnerbedarf Müller
Hopfenbuche	Grüner Finger
Baumwürger	Grüner Finger

Abbildung 10.31: Verknüpfte Tabellen: Zu allen Pflanzen wird der Firmenname des Lieferanten angezeigt.

Dieses `SELECT` lässt sich auch über eine alternative Syntax schreiben – über einen so genannten `INNER JOIN`:

```
SELECT name, firma
FROM pflanzen
INNER JOIN lieferanten
ON pflanzen.liefer_id = lieferanten.liefer_id;
```

Sie haben im Beispiel gesehen, wie Tabellen verknüpft werden. Das Feld `liefer_id` ist der Primärschlüssel in der Tabelle `lieferanten` und der Fremdschlüssel in der Tabelle `pflanzen`. Was passiert jetzt aber, wenn ein Lieferant gelöscht wird? Dann kann es sein, dass in der `pflanzen`-Tabelle noch ein Datensatz weiter existiert, der auf diesen Lieferanten verweist. Darum, dass Daten nicht inkonsistent werden, müssen Sie sich bei dem hier gewählten Tabellentyp MyISAM selbst kümmern. Beim InnoDB-Tabellentyp können Sie über so genannten FOREIGN KEY Constraints definieren, was in den einzelnen Fällen passieren soll (*http://dev.mysql.com/doc/refman/5.0/en/innodb-foreign-key-constraints.html*).

Tipp

Wenn Sie nicht angeben, wie zwei oder mehr Tabellen verknüpft werden sollen, erhalten Sie das Kreuzprodukt (kartesisches Produkt). Das bedeutet, jeder Datensatz der ersten Tabelle wird mit jedem Datensatz der zweiten Tabelle verknüpft. Das ist meistens kein erwünschtes Ergebnis.

```
SELECT name, firma
FROM pflanzen, lieferanten;
```

name	firma
Feldahorn	Gärtnerbedarf Müller
Feldahorn	Grünes Allerlei
Feldahorn	Grüner Finger
Warzenbirke	Gärtnerbedarf Müller
Warzenbirke	Grünes Allerlei
Warzenbirke	Grüner Finger
Pfeifenwinde	Gärtnerbedarf Müller
Pfeifenwinde	Grünes Allerlei
Pfeifenwinde	Grüner Finger
Filigranfarn	Gärtnerbedarf Müller
Filigranfarn	Grünes Allerlei
Filigranfarn	Grüner Finger
Kahle Apfelbeere	Gärtnerbedarf Müller
Kahle Apfelbeere	Grünes Allerlei
Kahle Apfelbeere	Grüner Finger
Essigrose	Gärtnerbedarf Müller
Essigrose	Grünes Allerlei
Essigrose	Grüner Finger
Büffelbeere	Gärtnerbedarf Müller
Büffelbeere	Grünes Allerlei
Büffelbeere	Grüner Finger
Hopfenbuche	Gärtnerbedarf Müller
Hopfenbuche	Grünes Allerlei
Hopfenbuche	Grüner Finger
Baumwürger	Gärtnerbedarf Müller
Baumwürger	Grünes Allerlei
Baumwürger	Grüner Finger

Abbildung 10.32: Kreuzprodukt – alle möglichen Kombinationen

10.10.1 Weitere Beispiele für Abfragen über mehrere Tabellen

Neben dem `INNER JOIN` werden Sie mitunter noch den `LEFT JOIN` und den `RIGHT JOIN` brauchen. Um deren Funktion zu sehen, sollten wir die Tabelle einmal modifizieren. Es kann ja jetzt sein, dass wir in der `pflanzen`-Tabelle eine Pflanze haben, für die noch kein Lieferant feststeht. Umgekehrt kann es auch einen Lieferanten geben, der derzeit noch keine Pflanzen liefert.

Ein neuer Lieferant wird eingefügt:

Listing 10.6: Ein weiterer Lieferant wird ergänzt (tabelle_lieferant_lieferantergaenzen.sql).

```
INSERT INTO lieferanten (liefer_id, firma, strasse, plz, ort)
VALUES (NULL, 'Gartenhandel Miu', 'Büchnerstraße 7', '12300', 'Unteroberübermoos');
```

Und ändern wir die Pflanzentabelle einmal so ab, dass beim Baumwürger kein Lieferant angegeben ist:

```
UPDATE pflanzen SET liefer_id = NULL WHERE pfl_id = 9;
```

Tipp

Übrigens können Sie sich über `SELECT LAST_INSERT_ID();` den zuletzt in die ID-Spalte eingefügten Autoinkrement-Wert ausgeben lassen.

Wenn Sie jetzt noch einmal zu allen Pflanzen die Lieferantennamen anzeigen lassen

```
SELECT name, firma
FROM pflanzen, lieferanten
WHERE pflanzen.liefer_id = lieferanten.liefer_id;
```

ist das Ergebnis dasselbe wie vor dem Einfügen der beiden Datensätze (siehe Abbildung 10.31). Auch ein `INNER JOIN` liefert dieses Ergebnis: Sie sehen nur Pflanzen, zu denen es Lieferanten gibt und ebenso nur Lieferanten, die auch eine Pflanze liefern.

Sollen jetzt wirklich alle Pflanzen mit ihren Lieferanten angezeigt werden, also auch diejenigen, für die es keinen Lieferanten gibt, können Sie `LEFT JOIN` benutzen:

```
SELECT name, firma
FROM pflanzen
LEFT JOIN lieferanten
ON pflanzen.liefer_id = lieferanten.liefer_id;
```

`LEFT JOIN` liefert von der links stehenden Tabelle (hier steht sie eigentlich nicht links, sondern davor) alle Datensätze. Das heißt: Jetzt wird die Pflanze ohne Lieferant, der Baumwürger, auch mit aufgeführt.

name	firma
Feldahorn	Gärtnerbedarf Müller
Warzenbirke	Grünes Allerlei
Pfeifenwinde	Grünes Allerlei
Filigranfarn	Grüner Finger
Kahle Apfelbeere	Grüner Finger
Essigrose	Gärtnerbedarf Müller
Büffelbeere	Gärtnerbedarf Müller
Hopfenbuche	Grüner Finger
Baumwürger	*NULL*

Abbildung 10.33: Jetzt werden alle Pflanzen angezeigt.

Ändern Sie das `LEFT JOIN` in ein `RIGHT JOIN`, erhalten Sie alle Datensätze der rechts stehenden Tabelle, d. h. alle Lieferanten, auch wenn sie noch keine Pflanzen liefern.

```
SELECT name, firma
FROM pflanzen
RIGHT JOIN lieferanten
ON pflanzen.liefer_id = lieferanten.liefer_id;
```

name	firma
Feldahorn	Gärtnerbedarf Müller
Essigrose	Gärtnerbedarf Müller
Büffelbeere	Gärtnerbedarf Müller
Warzenbirke	Grünes Allerlei
Pfeifenwinde	Grünes Allerlei
Filigranfarn	Grüner Finger
Kahle Apfelbeere	Grüner Finger
Hopfenbuche	Grüner Finger
NULL	Gartenhandel Miu

Abbildung 10.34: Dieses Mal sind alle Lieferanten aufgeführt.

Selbstverständlich hätten Sie dieses Ergebnis auch durch ein `LEFT JOIN` mit vertauschten Tabellen erreicht:

```
SELECT name, firma
FROM lieferanten
LEFT JOIN pflanzen
ON pflanzen.liefer_id = lieferanten.liefer_id;
```

Tipp

Da die Namen der Spalten, über die die beiden Tabellen verknüpft werden, identisch sind, können Sie anstelle von `ON pflanzen.liefer_id = lieferanten.liefer_id` auch `USING (liefer_id)` benutzen:

```
SELECT name, firma
FROM pflanzen
RIGHT JOIN lieferanten
USING (liefer_id);
```

Ebenfalls sehr nützlich ist die `GROUP BY`-Klausel. Um zu ermitteln, wie viele Pflanzen die einzelnen Lieferanten liefern, müssen wir die Inhalte gruppieren:

```
SELECT liefer_id, COUNT(*)
FROM pflanzen
GROUP BY liefer_id;
```

liefer_id	COUNT(*)
NULL	1
1	3
2	2
3	3

Abbildung 10.35: Anzahl gelieferte Pflanzen pro Lieferant

Durch `GROUP BY liefer_id` erreichen wir, dass gleiche Lieferanten-IDs zusammengefasst werden.

Soll jetzt der Firmenname anstelle der Lieferanten-ID angezeigt werden, müssen die Tabellen wieder verknüpft werden.

```
SELECT firma, COUNT(*) AS Anzahl
FROM pflanzen, lieferanten
WHERE pflanzen.liefer_id = lieferanten.liefer_id
GROUP BY pflanzen.liefer_id;
```

firma	Anzahl
Gärtnerbedarf Müller	3
Grünes Allerlei	2
Grüner Finger	3

Abbildung 10.36: Name des Lieferanten mit Anzahl an Pflanzen, die er liefert

Jetzt werden aber natürlich wieder nur die Datensätze berücksichtigt, die sich durch die `liefer_id` verknüpfen lassen.

Mit HAVING können Sie dann noch eine Bedingung formulieren, so werden durch folgende Abfrage nur die Lieferanten ausgegeben, die mehr als 2 Pflanzen liefern.

```
SELECT firma, COUNT(*) AS Anzahl
FROM pflanzen, lieferanten
WHERE pflanzen.liefer_id = lieferanten.liefer_id
GROUP BY pflanzen.liefer_id
HAVING Anzahl > 2;
```

firma	Anzahl
Gärtnerbedarf Müller	3
Grüner Finger	3

Abbildung 10.37: Nur die Lieferanten, die mehr als 2 Pflanzen liefern

10.11 Inhalte exportieren und importieren

Normalerweise werden Sie Ihre Webprojekte lokal entwickeln und testen. Im letzten Schritt geht es dann daran, diese auf den Webspace beim Provider zu laden. Bei den PHP-Skripten ist das soweit kein Problem, diese laden Sie per FTP hoch. Aber was ist mit der Datenbank? Da geht das anders.

Ihre Datenbank können Sie über phpMyAdmin exportieren. Wählen Sie zuerst einmal im linken Bereich die Datenbank aus und klicken Sie auf EXPORTIEREN.

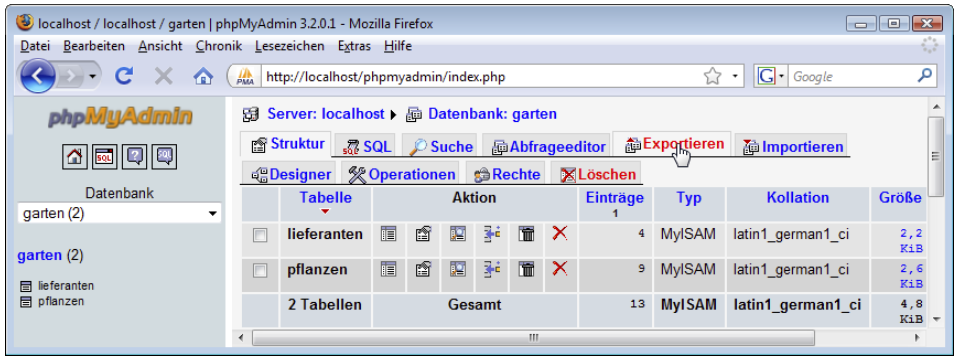


Abbildung 10.38: Exportieren wählen in phpMyAdmin

Im nun erscheinenden Fenster bestimmen Sie oben links, welche Tabellen Ihrer Datenbank Sie exportieren möchten. Aus den vielen Exportformaten wählen Sie SQL. Im rechten Bereich sollte sowohl STRUKTUR als auch DATEN ausgewählt werden.

Bei STRUKTUR lassen sich weitere Optionen festlegen. Über die ersten beiden bestimmen Sie, was geschehen soll, falls auf dem Zielsystem bereits eine gleichnamige Tabelle existiert:

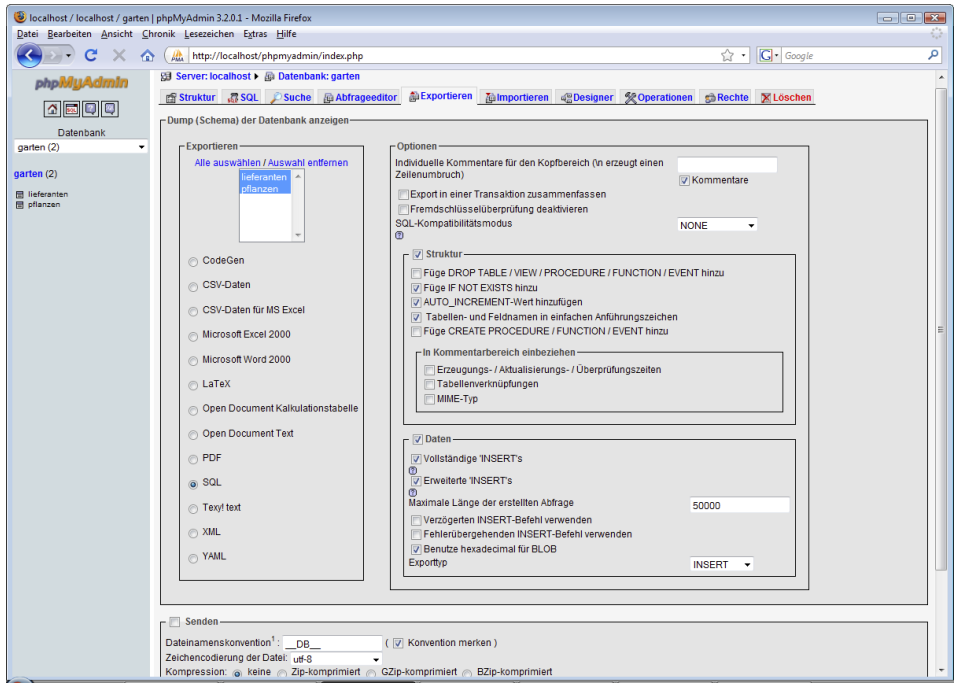


Abbildung 10.39: Exportoptionen

- FÜGE DROP TABLE / VIEW / PROCEDURE / FUNCTION HINZU bedeutet, dass eventuell vorhandene Tabellendefinitionen ersetzt werden.
- FÜGE IF NOT EXISTS HINZU lässt die ursprünglich auf dem Zielsystem vorhandenen Definitionen unverändert.

Die anderen Optionen können Sie auf dem Standard belassen. Wenn Sie dann auf OK klicken, werden Ihnen die MySQL-Befehle angezeigt, die zum Erstellen der Tabellen und zum Einfügen der Daten dienen. Dies können Sie nun kopieren und in einem Texteditor abspeichern.

Dann können Sie Tabellendefinition und Daten auf dem Zielsystem wieder importieren. Zuerst benötigen Sie eine Datenbank. Rufen Sie diese in phpMyAdmin auf oder erstellen Sie eine nach Bedarf. Klicken Sie dann auf IMPORTIEREN und wählen Sie die Datei aus. Als Typ geben Sie SQL an.

Tipp

Beim Exportieren oder Importieren von sehr großen Tabellen kann es Probleme geben, wenn der Import/Export-Vorgang länger dauert, als PHP-Skripten Zeit zur Ausführung gelassen wird. Verantwortlich dafür ist die Einstellung `max_execution_time` in der *php.ini*-Datei. Wenn Sie auf die Konfiguration Zugriff haben, können Sie den Wert heraufsetzen. Ansonsten sollten Sie die zu importierenden/exportierenden Daten portionieren, d. h. in kleinere Häppchen aufteilen.

Außerdem gibt es spezielle Tools, die für diese Aufgabe geeignet sind wie beispielsweise *Mysql dumper* (*http://www.mysqldumper.de/*) oder *Heidisql* (*http://www.heidisql.com/*).

Nach diesen MySQL-Grundlagen erfahren Sie im nächsten Kapitel, wie Sie mit PHP auf Ihre MySQL-Tabellen zugreifen.

11 PHP und MySQL

Nachdem Sie im letzten Kapitel wichtige MySQL-Grundlagen gelernt haben, sehen Sie nun, wie Sie mit PHP MySQL-Datenbanken ansprechen, um Daten anzeigen zu lassen, einzufügen, zu verändern oder zu löschen.

11.1 MySQLi – die verbesserte Erweiterung für MySQL

Zur Zusammenarbeit zwischen PHP und MySQL stehen zwei Erweiterungen zur Verfügung: MySQL und MySQLi. MySQLi steht für MySQL Improved Extension und ist die neuere Erweiterung. Sie erlaubt zwei Arten der Programmierung: prozedural und objektorientiert. Außerdem bietet sie attraktive Features wie Prepared Statements. Deswegen wird hier MySQLi verwendet.

Tipp

Ob die MySQLi-Erweiterung installiert ist, prüfen Sie wieder in der Ausgabe eines `phpinfo()`-Skripts. Suchen Sie nach *mysqli*.

MySQLi können Sie objektorientiert oder prozedural verwenden. Hier soll die objektorientierte Vorgehensweise gezeigt werden. Ein Beispiel für die prozedurale Programmierweise mit MySQLi sowie ein weiteres für die Verwendung der Schnittstelle MySQL finden Sie in Abschnitt 11.6.

Die Schnittstelle MySQLi besteht aus drei Klassen.

- `mysqli` ist für den Verbindungsaufbau zuständig.
- `mysqli_result`-Objekte enthalten das Ergebnis von `SELECT`-Abfragen.
- `mysqli_stmt` können Sie für vorbereitete Anweisungen (prepared statements) benutzen. Mehr dazu in Abschnitt 11.5.

11.1.1 MySQLi verwenden

Mehrere Schritte sind notwendig, um per PHP das Ergebnis einer MySQL-Abfrage auszugeben.

1. Zuerst müssen Sie eine Verbindung zum Datenbankserver erstellen. Diesen Schritt hatten Sie übrigens bei der Verwendung von phpMyAdmin nicht gesehen, da das phpMyAdmin für Sie erledigt hatte.
2. Außerdem müssen Sie die Datenbank auswählen, auf die Sie zugreifen möchten. Dies kann bei MySQLi aber auch in einem Schritt mit der Verbindung zum Datenbankserver erfolgen.
3. Dann müssen Sie die Abfrage an die Datenbank senden.
4. Als Nächstes muss das Ergebnis abgearbeitet und für die Webseite aufbereitet werden.
5. Am Ende schließen Sie die Verbindung.

Nun setzen wir die einzelnen Schritte per PHP um.

Tipp

Um die Beispiele in diesem Kapitel nachzuvollziehen, brauchen Sie die Datenbank `garten`, die im letzten Kapitel erstellt und mit zwei Tabellen befüllt wurde. Sie finden den vollständigen Code zur Erstellung der beiden Tabellen auch in der Datei *tabellen_struktur_daten.sql* im Ordner des aktuellen Kapitels bei den Listings dieses Buchs.

Schritt 1 & 2: Verbindung erstellen und Datenbank auswählen

Ein erstes kleines Skript versucht, die Verbindung zum Datenbankserver herzustellen und gibt im Erfolgsfall sowie auch bei Fehlern eine entsprechende Meldung aus:

```
$mysqli = new mysqli("localhost", "root", "geheim", "garten");
if ($mysqli->connect_error) {
  echo "Fehler bei der Verbindung: " . mysqli_connect_error();
  exit();
}
echo "Verbindung hat geklappt";
$mysqli->close();
```

Als Erstes erstellen Sie ein neues Objekt mit `new mysqli()`. In Klammern übergeben Sie zuerst den Namen des Datenbankservers, dann den Benutzernamen, das Passwort und den Namen der Datenbank, auf die Sie zugreifen möchten. Dies müssen Sie an Ihre Umgebung anpassen.

Tipp

Meist werden Sie für den Datenbankserver *localhost* wie im Beispiel verwenden. *localhost* steht dabei für das momentan genutzte System. Falls Sie hier etwas anderes bei Ihrem Hoster benutzen müssen, so erfahren Sie das von Ihrem Hoster.

Sie erhalten dann ein Objekt zurückgeliefert, das die Verbindung zur Datenbank darstellt. Sie erhalten aber ebenfalls ein Objekt zurück, wenn die Verbindung nicht geklappt hat. Deswegen wird zur Überprüfung, ob alles geklappt hat, auf die Eigenschaft `connect_error` des `mysqli`-Objekts zurückgegriffen. Dieses beinhaltet eine Beschreibung des letzten Verbindungsfehlers. Ist ein Fehler aufgetreten, wird dieser ausgegeben und das Skript mit `exit()` beendet. Ansonsten erscheint die Meldung »Verbindung hat geklappt« und die Verbindung wird über die `close()`-Methode geschlossen.

Schritt 3: Abfrage durchführen

Wenn die Verbindung steht, können Sie eine Abfrage durchführen. Diese schicken Sie über die `query()`-Methode des `mysqli`-Objekts an die Datenbank.

```
$ergebnis = $mysqli->query("SELECT name, beschreibung, preis FROM pflanzen;");
```

Als Ergebnis erhalten Sie ein Objekt vom Typ `mysqli_result`.

Schritt 4: Ergebnis für die Ausgabe aufbereiten

Das `mysqli_result`-Objekt bietet wiederum mehrere Methoden: Mit der Methode `fetch_array()` holen Sie sich den ersten Datensatz des Ergebnisses in Form eines Arrays.

```
$zeile = $ergebnis->fetch_array();
```

Dieses Array kann für einen ersten Überblick mit `print_r()` ausgegeben werden. Damit die Ausgabe von `print_r()` auch im Browser übersichtlich ist, wird das Element (X)HTML-Element `pre` eingesetzt. Danach wird die Methode `close()` für das `mysqli_result`-Objekt aufgerufen und damit der Speicherplatz freigegeben, den das Objekt belegt hat.

```
echo "<pre>";
print_r($zeile);
echo "</pre>";
$ergebnis->close();
```

Schritt 5: Verbindung schließen

Am Schluss sollten Sie die Verbindung wieder schließen:

```
$mysqli->close();
```

Und alles zusammen

Hier sehen Sie noch einmal die einzelnen Schritte zusammen:

Listing 11.1: Eine Abfrage wird durchgeführt und der erste Datensatz als Array ausgegeben (db_abfrage_ausfuehren.php).

```
01 $mysqli = new mysqli("localhost", "root", "geheim", "garten");
02 if ($mysqli->connect_error) {
03   echo "Fehler bei der Verbindung: " . mysqli_connect_error();
04   exit();
05 }
06 $ergebnis = $mysqli->query("SELECT name, beschreibung, preis FROM pflanzen;");
07 $zeile = $ergebnis->fetch_array();
08 echo "<pre>";
09 print_r($zeile);
10 echo "</pre>";
11 $ergebnis->close();
12 $mysqli->close();
```

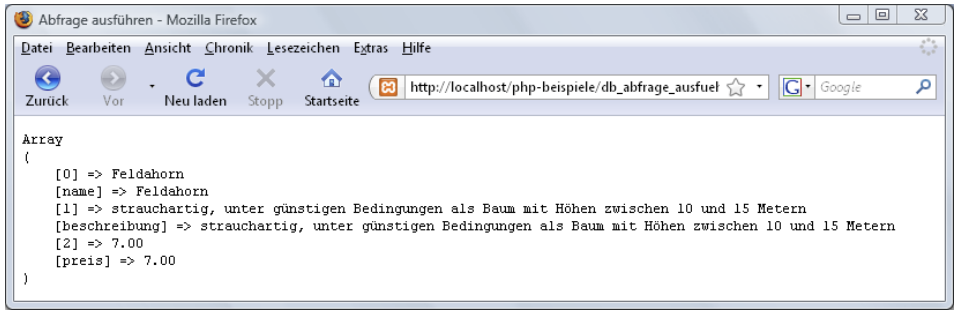


Abbildung 11.1: Ein Array mit dem ersten Datensatz

Ergebnis besser aufbereiten

In Abbildung 11.1 sehen Sie, dass die Methode `fetch_array()` ein Array mit dem Inhalt *des ersten Datensatzes* liefert. Das Array enthält als Schlüssel einerseits Zahlen und andererseits die Spaltennamen. Das heißt, Sie können auf den Preis der ersten Pflanze sowohl über `$zeile[2]` als auch über `$zeile["preis"]` zugreifen.

Jedoch hat `fetch_array()` uns nur den ersten Datensatz geliefert. Rufen Sie `fetch_array()` erneut auf, erhalten Sie den nächsten Datensatz als Array zurückgeliefert. Das geht so lange, wie Datensätze vorhanden sind. Deswegen bietet es sich an,

diese Funktion innerhalb eines Bedingungsblocks einer `while`-Schleife aufzurufen. Gibt es keinen Datensatz mehr, liefert `fetch_array()` `NULL` zurück und die Schleife ist damit beendet.

```
while($zeile = $ergebnis->fetch_array()) {
  echo "<strong>{$zeile['name']}</strong>: {$zeile['beschreibung']}
{$zeile['preis']}<br />\n";
}
```

Damit sieht das Beispiel folgendermaßen aus:

Listing 11.2: Alle Datensätze werden in einer Schleife ausgegeben (db_ausgabe_schleife.php).

```
01 mysqli = new mysqli("localhost", "root", "geheim", "garten");
02 if ($mysqli->connect_error) {
03   echo "Fehler bei der Verbindung: " . mysqli_connect_error();
04   exit();
05 }
06 $ergebnis = $mysqli->query("SELECT name, beschreibung, preis FROM pflanzen;");
07 while($zeile = $ergebnis->fetch_array()) {
08   echo "<strong>{$zeile['name']}</strong>: {$zeile['beschreibung']}
{$zeile['preis']}<br />\n";
09 }
10 $ergebnis->close();
11 $mysqli->close();
```

Zur Hervorhebung wird der Name der Pflanze in einem (X)HTML-`strong`-Element eingeschlossen.

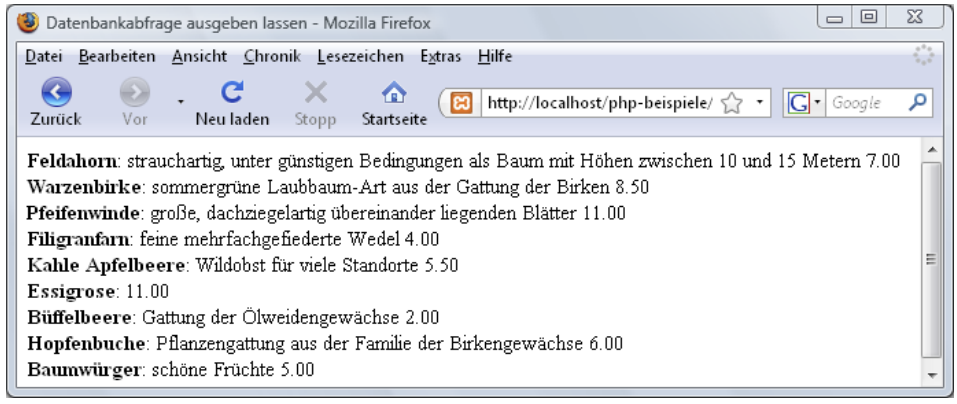


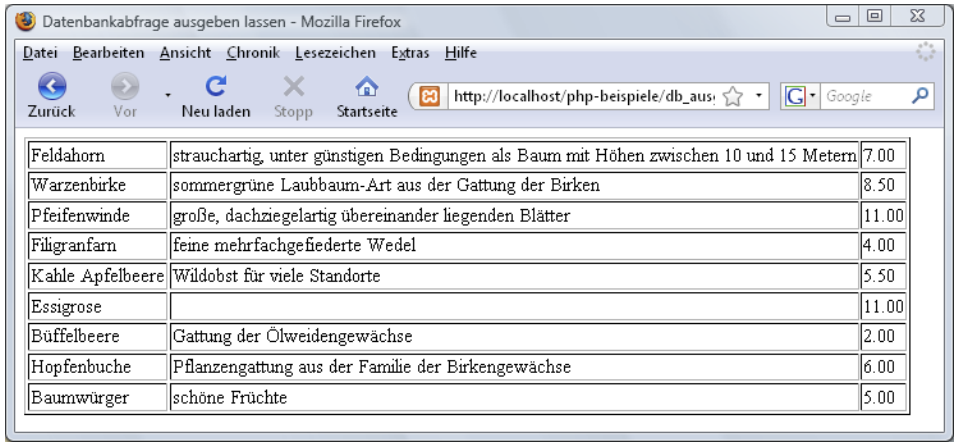
Abbildung 11.2: Alle Datensätze werden ausgegeben.

Lässt man viele Daten ausgeben, bietet sich zur Darstellung eine Tabelle an. Wie das geht, zeigt Listing 11.3. Dabei gibt es noch eine weitere Ergänzung. An sich könnten innerhalb der Texte (X)HTML-Tags enthalten sein, die die Anzeige empfindlich stören – oder noch Schlimmeres bewirken (vgl. Kapitel 7). Deswegen sollten Sie sicherheitshalber die auszugebenden Daten immer mit `htmlspecialchars()` vorbehandeln.

Listing 11.3: Ergebnis als Tabelle ausgeben lassen (db_ausgabe_tabelle.php)

```
01 $mysqli = new mysqli("localhost", "root", "geheim", "garten");
02 if ($mysqli->connect_error) {
03   echo "Fehler bei der Verbindung: " . mysqli_connect_error();
04   exit();
05 }
06 $ergebnis = $mysqli->query("SELECT name, beschreibung, preis FROM pflanzen;");
07 echo "<table border='1'>\n";
08 while($zeile = $ergebnis->fetch_array()) {
09   echo "<tr><td>" . htmlspecialchars($zeile["name"]) . "</td>"
10        . "<td>" . htmlspecialchars($zeile["beschreibung"]) . "</td>"
11        . "<td>" . htmlspecialchars($zeile["preis"]) . "</td>"
12        . "</tr>\n";
13 }
14 echo "</table>";
15
16 $ergebnis->close();
17 $mysqli->close();
```

Im Beispiel steht das `table`-Start-Tag vor Beginn der Schleife (Zeile 7), und das Endtag nach Beendigung der Schleife in Zeile 14. Innerhalb der Schleife wird die Tabellenzeile (`tr`) mit den einzelnen Zellen (`td`) samt Inhalt ausgegeben.



Feldahorn	strauchartig, unter günstigen Bedingungen als Baum mit Höhen zwischen 10 und 15 Metern	7.00
Warzenbirke	sommergrüne Laubbaum-Art aus der Gattung der Birken	8.50
Pfeifenwinde	große, dachziegelartig übereinander liegenden Blätter	11.00
Filigranfarn	feine mehrfachgefiederte Wedel	4.00
Kahle Apfelbeere	Wildobst für viele Standorte	5.50
Essigrose		11.00
Büffelbeere	Gattung der Ölweidengewächse	2.00
Hopfenbuche	Pflanzengattung aus der Familie der Birkengewächse	6.00
Baumwürger	schöne Früchte	5.00

Abbildung 11.3: Ausgabe als Tabelle

Neben `fetch_array()` gibt es folgende weitere Methoden, um die Datensätze auszulesen:

- `fetch_assoc()` liefert den Datensatz nur in Form eines assoziativen Arrays.
- `fetch_row()` liefert den Datensatz nur in Form eines indizierten Arrays.

`fetch_array()` macht also das, was `fetch_assoc()` und `fetch_row()` zusammen machen.

- `fetch_object()` liefert einen Datensatz als Objekt zurück. Die einzelnen Spalten sind die Eigenschaften und die Werte der Eigenschaften sind die Inhalte der Felder. Die `while`-Schleife zur Ausgabe lässt sich mit `fetch_object()` so realisieren:

Listing 11.4: Einsatz von fetch_object() (db_ausgabe_fetch_object.php)

```
while ($objekt = $ergebnis->fetch_object()) {
  echo "{$objekt->name}: {$objekt->beschreibung} {$objekt->preis}<br />\n";
}
```

11.2 Nützliche Informationen über das Ergebnis

Es gibt weitere nützliche Eigenschaften und Methoden des `mysqli`- und des `mysqli_result`-Objekts, über die Sie Informationen über das Ergebnis erhalten.

11.2.1 mysqli-Klasse

- Die Eigenschaft `affected_rows` des `mysqli`-Objekts gibt die Anzahl der vom letzten MySQL-Befehl betroffenen Datensätze zurück. Dies ist besonders nützlich bei `INSERT`-, `DELETE`- oder `UPDATE`-Aktionen, um festzustellen, ob diese auch erfolgreich waren.
- `insert_id` ist ebenfalls eine Eigenschaft des `mysqli`-Objekts und liefert den zuletzt automatisch eingetragenen ID-Wert.
- `error` liefert, sofern vorhanden, die Fehlermeldung des letzten MySQL-Befehls,
- `errno` beinhaltet den numerischen Fehlercode des letzten MySQL-Befehls

Ein Beispiel, in dem beim MySQL-Befehl *absichtlich ein Fehler* eingebaut ist, demonstriert den Einsatz von `error` und `errno`.

Listing 11.5: Abfrage mit eingebautem Fehler (db_fehler_query.php)

```
01 require_once "db_daten.php";
02 $ergebnis = $mysqli->query("SELECT name as Pflanzenname beschreibung AS
Kurzbeschreibung, preis AS Nettopreis FROM pflanzen WHERE preis < 7;");
03 if (!$ergebnis) {
04   echo "Der folgende Fehler ist aufgetreten: <strong>"
05        . $mysqli->error
06        . ".</strong><br />\n Die Fehlernummer: "
07        . $mysqli->errno;
```

```
08 } else {
09   echo "hat geklappt";
10   $ergebnis->close();
11 }
12 $mysqli->close();
```

In diesem Beispiel sehen Sie, dass der Code für die Verbindung ausgelagert ist und hier per `require_once` eingebunden wird (Zeile 1). Die Abfrage in Zeile 2 ist absichtlich fehlerhaft, es fehlt ein Komma zwischen `Pflanzenname` und `beschreibung`. In diesem Fall schlägt die Methode `query()` fehl und gibt `false` zurück. In der Bedingung von `if` wird `$ergebnis` geprüft. Hier ist `!$ergebnis` wahr und es kann auf die Fehlermeldung zugegriffen und diese inklusive Nummer des Fehlercodes ausgegeben werden.

Im `else`-Zweig (ab Zeile 8) wird in Listing 11.5 nur eine Erfolgsmeldung ausgegeben und die Ressource wieder freigegeben – hier würde man normalerweise die Verarbeitung unterbringen, d. h. das Ergebnis in Form von einer Tabelle ausgeben lassen.

Abbildung 11.4: Die durch die falsche QUERY ausgelöste Fehlermeldung

Tipp

Wenn Sie nicht überprüfen, ob `query()` geklappt hat und danach eine Methode wie `fetch_array()` auf das `$ergebnis` anwenden, erhalten Sie die in Abbildung 11.5 gezeigte Fehlermeldung.

Abbildung 11.5: Fehlermeldung beim Versuch der Weitbildrverarbeitung einer fehlerhaften Abfrage

Denn dann enthält `$ergebnis` den Wert `false` und ist kein Objekt, deswegen ergibt der Einsatz von `fetch_array()` die entsprechende Fehlermeldung.

11.2.2 mysqli_result-Klasse

Auf weitere Informationen über das Ergebnis können Sie über das `mysqli_result`-Objekt zugreifen.

- `field_count`: Über diese Eigenschaft lesen Sie die Anzahl der Spalten aus.
- `num_rows`: Diese Eigenschaft liefert die Anzahl der Datensätze eines Ergebnissatzes.
- `fetch_fields()`: Diese Methode liefert Ihnen nützliche Informationen zu den einzelnen Spalten als Objekt. Sie können den Namen bei Spalten, sowie den ursprünglichen Namen bei der Verwendung eines Aliases, den Namen der Tabelle und die maximale Zeichenzahl ausgeben lassen.

Listing 11.6: Informationen ausgeben lassen (db_infos.php)

```
01 require_once "db_daten.php";
02 $ergebnis = $mysqli->query("SELECT name as Pflanzenname, beschreibung AS
     Kurzbeschreibung, preis AS Nettopreis FROM pflanzen WHERE preis < 7;");
03 if ($ergebnis) {
04   $spaltenzahl = $ergebnis->field_count;
05   $zeilenzahl = $ergebnis->num_rows;
06   $betroffen = $mysqli->affected_rows;
07
08   echo "<p>Insgesamt $zeilenzahl Datensätze gefunden</p>\n";
09   echo "<p>Spaltenanzahl: $spaltenzahl</p>\n";
10   echo "<p>Betroffene Datensätze: $betroffen</p>\n";
11
12   echo "<h2>Informationen zu den einzelnen Spalten</h2>\n";
13   $infos = $ergebnis->fetch_fields();
14   foreach($infos as $inf) {
15     echo "<p>Name: " . $inf->name . "<br />\n";
16     echo "Urspr. Name: " . $inf->orgname . "<br />\n";
17     echo "Tabelle: " . $inf->table . "<br />\n";
18     echo "Längste Zeichenkette: " . $inf->max_length . "</p>\n";
19   }
20   $ergebnis->close();
21 }
22 $mysqli->close();
```

11.3 MySQL-Sonderzeichen behandeln

Wenn Sie Daten in eine MySQL-Datenbank eintragen oder Abfragen formulieren, müssen Sie Zeichen, die in MySQL eine spezifische Bedeutung haben, maskieren, d. h. schützen. Wie das geht und warum das so ist, soll anhand eines Beispiels mit dem `INSERT`-Befehl gezeigt werden. Die Notwendigkeit der Behandlung von Sonderzeichen gilt aber gleichermaßen für andere MySQL-Befehle wie `SELECT` oder `UPDATE`.

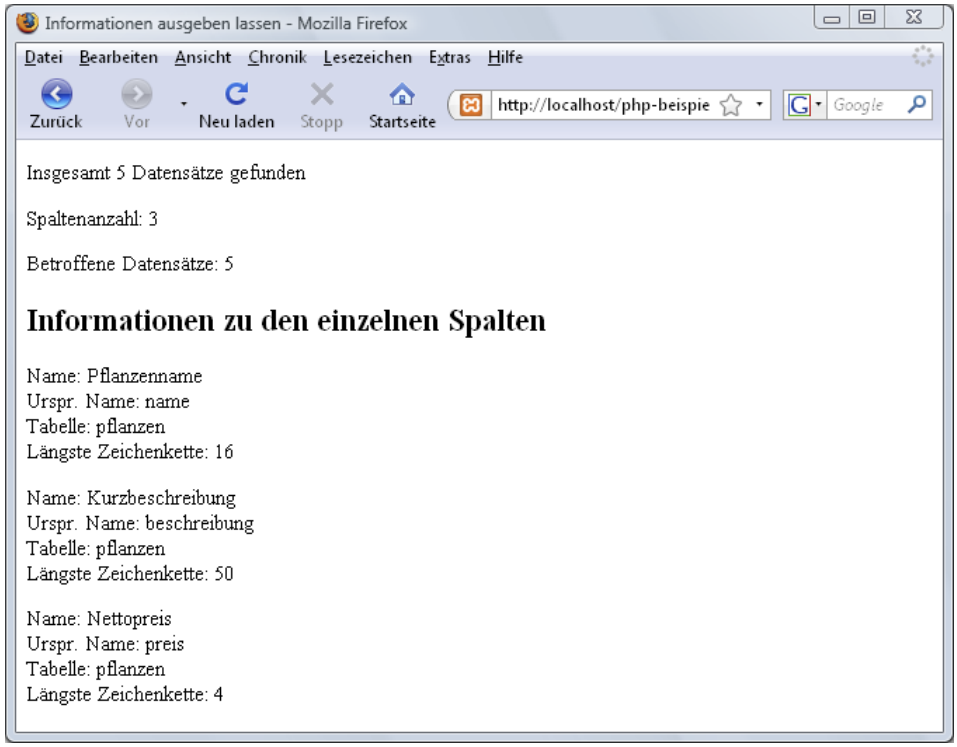


Abbildung 11.6: Informationen zum Ergebnis ausgeben lassen

Wollten wir beispielsweise einen Lieferanten mit dem Firmennamen *O'Briens Gartenbedarf* eintragen, so bekommen wir Probleme:

```
INSERT INTO lieferanten VALUES (NULL, 'O'Briens Gartenbedarf', 'Holzweg 2',
'12345', 'Dort');
```

Denn wenn gleichzeitig die einfachen Anführungszeichen als Begrenzungszeichen für Strings verwendet werden, stoßen die sich mit dem Apostroph bei O'Brien. Dieser Apostroph muss geschützt, d. h. durch einen Backslash maskiert werden:

```
INSERT INTO lieferanten VALUES (NULL, 'O\'Briens Gartenbedarf', 'Holzweg 2', '12345',
'Dort');
```

Welche Zeichen jeweils maskiert werden müssen, kann in den einzelnen Datenbankmanagementsystemen unterschiedlich sein. Deswegen nehmen Sie am besten zum »Escapen« eine Funktion, die speziell auf das jeweilige System – hier MySQL – zugeschnitten ist. Für die MySQLi-Erweiterung lautet die entsprechende Methode `real_escape_string()`. Sie ist eine Methode des `mysqli`-Objekts, erwartet einen String als Parameter und liefert einen String zurück:

```
$firma = $mysqli->real_escape_string($firma);
```

Achtung

Falls die Datenbankverbindung nicht geklappt hat, liefert `real_escape_string()` einen leeren String zurück.

Ein Beispiel demonstriert den `INSERT`-Vorgang unter Einsatz von `real_escape_string()`:

Listing 11.7: Datensatz eintragen (db_insert_escapen.php)

```
01 require_once "daten.php";
02 $firma   = "O'Briens Gartenbedarf";
03 $strasse = "Holzweg 2";
04 $plz     = "12345";
05 $ort     = "Dort";
06
07 $firma   = $mysqli->real_escape_string($firma);
08 $strasse = $mysqli->real_escape_string($strasse);
09 $plz     = $mysqli->real_escape_string($plz);
10 $ort     = $mysqli->real_escape_string($ort);
11 $insert  = "INSERT INTO lieferanten
12            (firma, strasse, plz, ort)
13            VALUES ('$firma', '$strasse', '$plz', '$ort')";
14 echo $insert;
15 if($ergebnis = $mysqli->query($insert)) {
16   echo "<br />\nAnzahl der veränderten Datensätze: "
17        . $mysqli->affected_rows;
18 } else {
19   echo $mysqli->error;
20 }
21 $mysqli->close();
```

Abbildung 11.7: So klappt das Einfügen: Beim MySQL-Befehl sieht man, dass der Apostroph wie notwendig maskiert wurde.

In der Datenbanktabelle werden dann die Inhalte korrekt eingefügt und ohne Backslash gespeichert.

Falls jedoch die Daten, die Sie eintragen möchten, aus einem Formular stammen und die Konfigurationseinstellung `MAGIC_QUOTES` (siehe Kapitel 7) aktiviert ist, erhalten diese schon automatisch einen Backslash. Dies sehen Sie am nächsten Beispiel.

Listing 11.8: Datensatz aus Formular in Datenbank schreiben (db_insert_formular.php)

```
<?php
if (empty($_POST["firma"])) {
?>
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]) ?>" method="post">
Firma: <br /><input type="text" name="firma" /><br />
Straße: <br /><input type="text" name="strasse" /><br />
PLZ: <br /><input type="text" name="plz" /><br />
Ort: <br /><input type="text" name="ort" /><br />
<input type="submit" name="abgeschickt"/>
</form>
<?php
} else {
  require_once "db_daten.php";
  $firma   = $_POST["firma"];
  $strasse = $_POST["strasse"];
  $plz     = $_POST["plz"];
  $ort     = $_POST["ort"];

  $firma   = $mysqli->real_escape_string($firma);
  $strasse = $mysqli->real_escape_string($strasse);
  $plz     = $mysqli->real_escape_string($plz);
  $ort     = $mysqli->real_escape_string($ort);
  $insert = "INSERT INTO lieferanten
            (firma, strasse, plz, ort)
             VALUES ('$firma', '$strasse', '$plz', '$ort')";
  echo $insert;
  if($ergebnis = $mysqli->query($insert)) {
    echo "<br />\nAnzahl der veränderten Datensätze: "
         . $mysqli->affected_rows;
  } else {
    echo $mysqli->error;
  }
  $mysqli->close();
}
?>
```

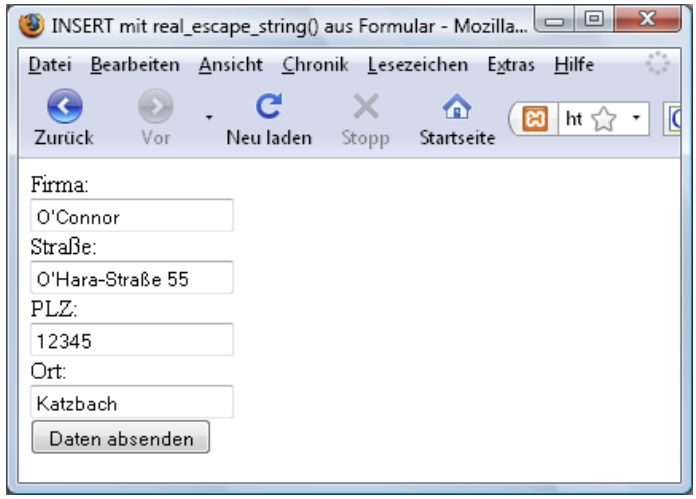


Abbildung 11.8: Der Beispieldatensatz mit Apostroph, der eingetragen werden soll

In Zeile 2 wird überprüft, ob das Formularfeld mit Namen `firma` ausgefüllt wurde, wenn nein, wird das Formular ausgegeben. Ansonsten findet ab Zeile 12 die Bearbeitung der Formulardaten statt: Sie werden mit `real_escape_string()` behandelt (ab Zeile 19). In Zeile 23 wird der `INSERT`-String zusammengesetzt und zur Veranschaulichung in Zeile 26 ausgegeben. In Zeile 27 wird der MySQL-Befehl an die Datenbank gesendet und in Zeile 29 die Anzahl der geänderten Zeilen ausgegeben.

Gibt man jetzt in das erstellte Formular Inhalte mit einfachen Apostrophen ein (Abbildung 11.9), erhält man bei aktivierten Magic Quotes nicht das gewünschte Ergebnis (Abbildung 11.10).

Abbildung 11.9: Eindeutig zu viele Backslashes ...

Im Beispiel wird der `INSERT`-Befehl zu Testzwecken wieder ausgegeben. Sie sehen drei Backslashes vor den Apostrophen. Durch die `MAGIC QUOTES` wird einer eingefügt: *O\'Connor*. `real_escape_string()` ergänzt dann zwei weitere für die zwei Sonderzeichen – für den Apostroph und für den Backslash selbst.

Beim entsprechenden Datensatz in phpMyAdmin sieht man den Backslash als integralen Bestandteil des Inhalts.

O\'Connor	O\'Hara-Straße 55	12345	Katzbach

Abbildung 11.10: Nicht erwünscht: Die Backslashes sind jetzt so in der Tabelle eingetragen.

Das ist so eindeutig nicht erwünscht. Um das zu verhindern, gibt es mehrere Möglichkeiten:

- Die beste Option ist, `MAGIC QUOTES` zu deaktivieren (siehe Kapitel 7) und alle Daten, die Sie in eine Datenbanktabelle schreiben, mit `real_escape_string()` zu bearbeiten oder die in Abschnitt 11.5 erläuterten Prepared Statements zu benutzen, bei denen automatisch `real_escape_string()` auf die Daten angewendet wird. Dies ist die empfohlene Variante, die zukunftssicher ist, da es `MAGIC QUOTES` in PHP 6 nicht mehr geben wird.
- Falls Sie nicht auf die Konfiguration zugreifen können, können Sie die Backslashes auch entfernen wie in Kapitel 7 beschrieben und dann `real_escape_string()` benutzen.
- Sie können auch mit `get_magic_quots_gpc()` prüfen, ob `MAGIC QUOTES` aktiviert sind, und in diesem Fall die Behandlung mit `real_escape_string()` unterlassen.

Im Folgenden wird davon ausgegangen, dass Sie `MAGIC QUOTES` deaktiviert haben.

11.4 SQL-Injections

Dieses Maskieren von MySQL-Sonderzeichen ist ganz wichtig als Schutz vor so genannten SQL-Injections. Bei einer SQL-Injection versucht ein Angreifer, eigene Datenbankbefehle einzuschleusen. Die Ziele dabei sind, Zugriff auf geschützte Bereiche zu erhalten, weitere Daten von anderen Kunden zu sehen oder einfach nur Schaden anzurichten.

Hier einmal ein klassisches Beispiel für eine SQL-Injection. Nehmen wir an, es gibt eine Tabelle `benutzer`, die Benutzernamen mit zugehörigen Kennwörtern verwaltet. Die Benutzer müssen Benutzernamen und Passwort in ein Formular eintragen. Mit diesen Daten wird ein `SELECT` auf die Datenbank ausgeführt. Ist ein Datensatz zu der eingegebenen Kombination aus Benutzernamen und Passwort vorhanden, geht man davon aus, dass es sich um einen berechtigten Benutzer handelt. Nehmen wir weiter an, dass die Sonderzeichen weder über `MAGIC QUOTES` noch `real_escape_string()` geschützt werden.

Der anfällige `SELECT`-Befehl sieht nun so aus:

```
$abfrage = "SELECT id, name FROM benutzer WHERE name='{$_POST['name']}' AND
passwort='{$_POST['passwort']}'";
```

Hier werden die Daten aus dem Formular *direkt und ungefiltert* in den Query-String eingebaut. Gibt jemand beispielsweise `florence` und `geheim` ein, ergibt sich folgender Query-String:

```
SELECT id, name FROM passwort WHERE name='florence' AND passwort='geheim'
```

Wenn jemand jedoch in das Formularfeld für den Benutzernamen Folgendes eingibt:

```
florence' OR 1='1
```

ergibt sich folgender `SELECT`-Befehl:

```
SELECT id, name FROM passwort WHERE name='florence' OR 1='1' AND passwort=''
```

Und damit wird die ID zurückgegeben, obwohl kein Passwort eingegeben wurde, und jemand hat sich allein durch Kenntnis eines Benutzernamens Zugriff verschafft.

Werden hingegen die Sonderzeichen maskiert – z. B. durch `real_escape_string()` – funktioniert das Ganze nicht mehr:

```
SELECT id, name FROM passwort WHERE name='florence\' OR 1=\'1' AND passwort=''
```

Hinweis

Das war jetzt nur ein Beispiel für eine SQL-Injection, es gibt viele weitere und komplexere. Beispielsweise kann bei einer SQL-Injection versucht werden, einen zusätzlichen SQL-Befehl abzusenden, der den Inhalt einer Tabelle löscht etc. Das ist jedoch über die vorgestellten `mysqli`-Befehle nicht möglich, da darüber standardmäßig *nur ein* Befehl abgesendet werden kann, anders ist das hingegen bei der Methode `multi_query()` des `mysqli`-Objekts.

Den besten Schutz vor SQL-Injection, bieten die weiter unten vorgestellten Prepared Statements: Einerseits wird dabei automatisch `real_escape_string()` eingesetzt und andererseits ist dabei der MySQL-Befehl von den eingefügten Parametern logisch getrennt.

Achtung

Wo wir gerade beim Thema Sicherheit sind: Ihre Zugangsdaten zum Datenbankserver verdienen selbstverständlich besonderen Schutz. Speichern Sie die Zugangsdaten in einer eigenen Datei, die Sie per `require` einbinden. Außerdem sollten, sofern das bei Ihrem Provider möglich ist, die Zugangsdaten außerhalb des Document Root (bei XAMPP wäre das *htdocs*) des Webverzeichnisses speichern: Dann können diese nicht über das Internet angefordert werden. Steht diese Option nicht zur Verfügung, sollten Sie diese Daten zumindest in einem anderen Ordner abspeichern, den Sie über eine *.htaccess*-Datei (siehe Anhang) vor einem direkten Zugriff schützen.

11.5 Prepared Statements – auf alles bestens vorbereitet

Prepared Statements sind eine alternative Art, MySQL-Anweisungen an den Datenbankserver zu senden, die mehrere Vorteile bietet.

Prepared Statements sind so genannte vorbereitete Anweisungen, bei denen die Anweisung erst einmal noch keine Werte enthält. Stattdessen werden *Platzhalter* übergeben. Sie sind eine sichere Methode zur Verhinderung von SQL-Injections, da zum einen die Gültigkeit der Parameter vom Datenbanksystem vor der Verarbeitung überprüft wird und zum anderen bei Prepared Statements die Abfragestruktur von den Parametern getrennt ist. Zudem bieten Prepared Statements einen Geschwindigkeitsvorteil, wenn eine Anweisung mit unterschiedlichen Parametern mehrmals durchgeführt wird – also beispielsweise hintereinander mehrere gleich geformte `INSERT`s abgesetzt werden.

Bisher haben Sie mit zwei Klassen zu tun gehabt: `mysqli` und `mysqli_result`. Bei den vorbereiteten Anweisungen kommt eine neue Klasse zum Einsatz: `mysqli_stmt`. Ein erstes Beispiel zeigt, wie man eine vorbereitete Anweisung erstellt.

Listing 11.9: Eine vorbereitete Anweisung (db_vorbereitet_anweisung.php)

```
01 require_once "db_daten.php"
02 $name = "Schmetterlingsstrauch";
03 $beschreibung = "schöne Blüten";
04 $preis = 6;
05 if($stmt = $mysqli->prepare("INSERT INTO pflanzen
06                              (name, beschreibung, preis)
07                               VALUES (?, ?, ?)")) {
08   $stmt->bind_param("ssd", $name, $beschreibung, $preis);
09   $stmt->execute();
10   echo "Anzahl der veränderten Datensätze : "
11        . $stmt->affected_rows;
12   $stmt->close();
13 }
```

Wenn Sie die Methode `prepare()` bei einem `mysqli`-Objekt aufrufen, erhalten Sie ein `mysqli_stmt`-Objekt zurück, das sehen Sie in Zeile 5. Dabei erhält `$mysqli->prepare()` als Parameter die vorbereitete MySQL-Anweisung. Sie sieht aus wie gewohnt, enthält aber *Fragezeichen als Platzhalter* für die einzugebenden Daten.

Über die Methode `bind_param()` des `mysqli_stmt`-Objekts werden die Parameter mit PHP-Variablen verbunden. `$stmt->bind_param()` enthält zuerst einen Formatierungsstring, der die Datentypen der einzelnen Parameter bestimmt. Jeder Buchstabe steht dabei für einen Parameter eines bestimmten Datentyps. `"ssd"` bedeutet, dass es drei Parameter gibt, wobei die ersten beiden Strings sind und der dritte eine Fließkommazahl ist. Danach werden die Variablen mit den Inhalten übergeben.

Mit `$stmt->execute()` in Zeile 9 wird dann der MySQL-Befehl auch wirklich ausgeführt.

Anschließend wird über `$stmt->affected_rows` ermittelt, wie viele Datensätze geändert wurden und die Anzahl ausgegeben. Schließlich gibt `$stmt->close()`das `stmt`-Objekt wieder frei.

Nun noch einmal zum Formatierungsstring bei der vorbereiteten MySQL-Anweisung. Die Anzahl der Buchstaben im Formatierungsstring muss der Anzahl der übergebenden Parameter entsprechen. Hier ein paar weitere Beispiele für mögliche parametrisierte MySQL-Befehle:

Nur zwei Parameter werden übergeben:

```
$stmt = $mysqli->prepare("INSERT INTO pflanzen (name, preis)
VALUES (?, ?,)";
$stmt->bind_param("sd", $name, $preis);
```

`NULL` kann auch direkt übergeben werden:

```
$stmt = $mysqli->prepare("INSERT INTO pflanzen (pflanzen_id, name, preis)
VALUES (NULL, ?, ?,)";
$stmt->bind_param("sd", $name, $preis);
```

Welche Zeichen im Formatierungsstring angegeben werden können, zeigt Tabelle 11.1.

Zeichen	Bedeutung
`i`	Integer (ganze Zahlen)
`d`	Fließkommazahl
`b`	BLOB
`s`	Strings

Tabelle 11.1: Mögliche Datentypen in Prepared Statements

Achtung

Sie können beim Binden der Parameter über `bind_param()` immer nur Variablen übergeben, keine Strings! Das Folgende würde also nicht funktionieren:

```
$stmt->bind_param("sd", $name, 14);
```

Gerade haben Sie gesehen, wie Sie über vorbereitete Anweisungen ein `INSERT` durchführen, nun ein Beispiel für ein `SELECT`:

Listing 11.10: Einen SELECT-Befehl über eine vorbereitete Anweisung (db_vorbereitet_select.php)

```
01 require_once "db_daten.php";
02 $suche = "%beere%";
03 if($stmt = $mysqli->prepare("SELECT name, beschreibung, preis
04                              FROM pflanzen
05                              WHERE name LIKE ?")) {
06   $stmt->bind_param("s", $suche);
07   $stmt->execute();
08   $stmt->bind_result($name, $beschreibung, $preis);
09   echo "<table border='1'>\n";
10   while($stmt->fetch()) {
11     echo "<tr>\n\t<td>"
12        . htmlspecialchars($name)
13        . "</td>\n\t<td>"
14        . htmlspecialchars($beschreibung)
15        . "</td>\n\t<td>"
16        . htmlspecialchars($preis)
17        . "</td>\n</tr>\n";
18   }
19   $stmt->close();
20 }
21 $mysqli->close();
```

Ein `SELECT` funktioniert bei vorbereiteten Anweisungen ganz ähnlich wie ein `INSERT`. Zuerst wird eine Anweisung mit der `prepare()`-Methode vorbereitet und per `bind_param()` werden die Parameter gebunden. Im Beispiel wird ein `SELECT` mit `LIKE` benutzt. Sie sehen, dass bei der vorbereiteten Anweisung hinter `LIKE` nur das Fragezeichen steht. Die Platzhalter `%` werden bei der Variablen selbst eingefügt (Zeile 2).

Entscheidend ist jetzt Zeile 8, hier werden die Spalten des Ergebnisses an Variablen gebunden. Im Beispiel sind es drei Spalten (`name`, `beschreibung`, `preis`) und entsprechend werden auch drei Variablen bei `bind_result()` angegeben.

Nun kann man mit `fetch()` die Ergebnisse in einer `while`-Schleife durchlaufen und ausgeben.

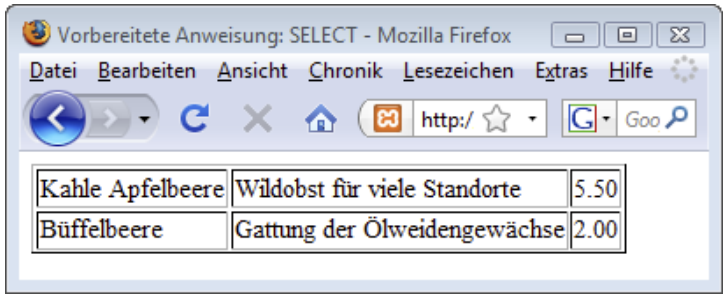


Abbildung 11.11: Das Ergebnis der Abfrage

Optional können Sie nach `$stmt->execute()` noch `$stmt->store_result()` aufrufen, dann wird das ganze Ergebnis zum Client übertragen – ansonsten verbleibt es beim Server und wird erst durch `fetch()` zum Client übertragen. Wenn Sie `$stmt->store_result()` verwenden, können Sie mit `$stmt->num_rows` die Anzahl an Zeilen des Ergebnisses ermitteln. Danach sollten Sie den Speicherplatz mit `free_result()` wieder freigeben.

Listing 11.11: Vorbereitete Anweisung mit store_result() (db_vorbereitet_store.php)

```
01 require_once "db_daten.php";
02 $suche = "%beere%";
03 if($stmt = $mysqli->prepare("SELECT name, beschreibung, preis
04                               FROM pflanzen
05                               WHERE name LIKE ?")) {
06   $stmt->bind_param("s",  $suche);
07   $stmt->execute();
08   $stmt->store_result();
09   $anzahl = $stmt->num_rows;
10   $stmt->bind_result($name, $beschreibung, $preis);
11   echo "<h2>Insgesamt $anzahl Ergebnisse für '$suche'</h2>\n";
12   echo "<table border='1'>\n";
13   while($stmt->fetch()) {
14     echo "<tr>\n\t<td>"
15        . htmlspecialchars($name)
16        . "</td>\n\t<td>"
17        . htmlspecialchars($beschreibung)
18        . "</td>\n\t<td>"
19        . htmlspecialchars($preis)
20        . "</td>\n</tr>\n";
21     }
22   $stmt->free_result();
23   $stmt->close();
24 }
25 $mysqli->close();
```

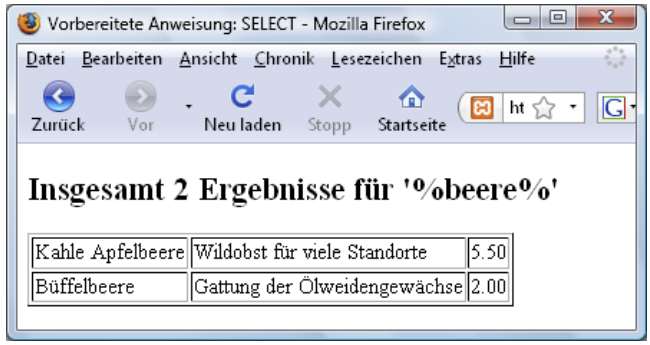


Abbildung 11.12: Das Ergebnis für die Abfrage – zusätzlich wird die Anzahl der Ergebnisse ausgegeben.

11.5.1 Daten über ein Formular eingeben, ändern und löschen

An einem Beispiel soll jetzt die Eingabe per Formular in die Datenbank noch einmal demonstriert werden, eingesetzt werden durchgängig Prepared Statements. Gezeigt wird ein Skript, das verwendet werden kann, um aktuelle Nachrichten online zu verwalten: Es können neue Nachrichten erstellt sowie bestehende verändert und gelöscht werden.

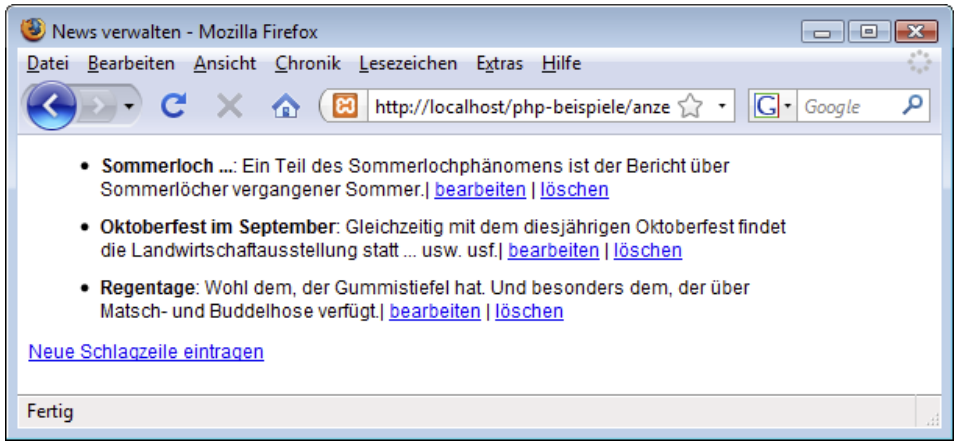


Abbildung 11.13: Die Anzeige aller Nachrichten inklusive jeweils eines Links zum Löschen und zum Bearbeiten

Erst einmal muss hierfür eine Tabelle angelegt werden. Dazu dient folgender MySQL-Befehl:

Listing 11.12: SQL-Befehle zur Erstellung der Tabelle aktuell (tabelle_aktuell_erstellen.sql)

```
CREATE TABLE aktuell (
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  titel VARCHAR(50) NOT NULL,
  text VARCHAR(250) NULL
) ENGINE=MyISAM;
```

Außerdem sollten Sie ein paar Datensätze über phpMyAdmin eintragen oder den folgenden Code im SQL-Fenster von phpMyAdmin eingeben:

Listing 11.13: Drei Datensätze für die Tabelle aktuell (tabelle_aktuell_daten.sql)

```
INSERT INTO aktuell (titel, text) VALUES
('Sommerloch ...', 'Ein Teil des Sommerlochphänomens ist der Bericht über
Sommerlöcher vergangener Sommer.'),
('Oktoberfest im September', 'Gleichzeitig mit dem diesjährigen Oktoberfest findet
```

```
die Landwirtschaftausstellung statt ... usw. usf.'),
('Regentage', 'Wohl dem, der Gummistiefel hat. Und besonders dem, der über Matsch-
und Buddelhose verfügt.');
```

Jetzt zur PHP-Programmierung. Im Beispiel werden vier Skripte zum Verwalten der Nachrichten benötigt. Im »echten« Einsatz würden Sie diese Skripten in einem eigenen Ordner unterbringen, der über eine *.htaccess*-Datei geschützt ist, d.h., nur nach Eingabe eines Passworts erreichbar ist. Schließlich sollen nur berechtigte Benutzer neue Nachrichten eintragen dürfen. Die Anzeige der Nachrichten wird hingegen in eine andere öffentlich zugängliche Seite integriert – aber ohne die Links zum Bearbeiten der Nachrichten.

Sehen wir uns jetzt an, welche vier Skripten zur Verwaltung der Inhalte benutzt werden:

- *anzeigen.php* stellt alle Nachrichten dar. Jede Nachricht ist außerdem mit einem Link zum LÖSCHEN und BEARBEITEN versehen, die zu den entsprechenden Skripten führen. Außerdem gibt es einen Link, um eine neue Nachricht zu erstellen.
- *neu.php* beinhaltet ein Formular zum Eintragen einer neuen Nachricht.
- *bearbeiten.php*: Dieses Skript wird über *anzeigen.php* aufgerufen und erlaubt es, ausgewählte Nachrichten zu bearbeiten: Diese werden in einem Formular angezeigt und können editiert werden.
- *loeschen.php* macht das, was der Name verspricht.

Skript zur Anzeige

Sehen wir uns zuerst das Skript zur Anzeige an:

Listing 11.14: Das Skript zur Ausgabe aller Nachrichten (anzeigen.php)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <title>News verwalten</title>
06   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
07   <style type="text/css">
08   body { font-size: 80%; font-family: sans-serif; }
09   ul { width: 40em; }
10   li { margin: 10px;}
11   </style>
12 </head>
13 <body>
14 <?php
15 require_once "db_daten_test.php";
16 if($stmt = $mysqli->prepare("SELECT id, titel, text FROM aktuell")) {
17   $stmt->execute();
```

```
18   $stmt->bind_result($id, $titel, $text);
19   echo "<ul>\n";
20   while($stmt->fetch()) {
21     echo "<li><strong>"
22          . htmlspecialchars($titel)
23          . "</strong>: "
24          . htmlspecialchars($text)
25          . "| <a href='bearbeiten.php?id="
26          . (int)$id
27          . "'>bearbeiten</a> "
28          . "| <a href='loeschen.php?id="
29          . (int)$id
30          . "'>löschen</a>"
31          . "</li>";
32   }
33   echo "</ul>\n";
34   $stmt->close();
35 }
36 $mysqli->close();
37 ?>
38 <a href="neu.php">Neue Schlagzeile eintragen</a>
39 </body>
40 </html>
```

Zu Beginn stehen der HTML-Header sowie ein paar minimale CSS-Formatierungen in den Zeilen 8–10.

Der PHP-Code beginnt in Zeile 14. Im Beispiel werden durchgängig Prepared Statements eingesetzt. Zeile 16 erstellt das Prepared Statement, in Zeile 17 wird es ausgeführt und in Zeile 18 die Spalten des Ergebnisses an Variablen gebunden. Ausgegeben werden die Nachrichten als ungeordnete Liste: Vor und nach der `while`-Schleife werden `<ul>` (Zeile 19) und `</ul>` (Zeile 33) ausgegeben. Innerhalb der `while`-Schleife werden die `li`-Elemente für die einzelnen Nachrichten geschrieben: Zuerst wird der Titel ausgegeben und dann der Text. Dahinter stehen die Links zum Bearbeiten und zum Löschen, über die die entsprechenden Skripte aufgerufen werden. Damit aber die richtige Nachricht bearbeitet oder gelöscht wird, muss die jeweilige `id` der Nachricht übergeben werden. Sehen wir uns das einmal genauer an:

```
25          . "| <a href='bearbeiten.php?id="
26          . (int)$id
27          . "'>bearbeiten</a>"
```

Damit wird ein Link zusammengestellt, der beispielsweise so aussieht:

```
<a href='bearbeiten.php?id=1'>bearbeiten</a>
```

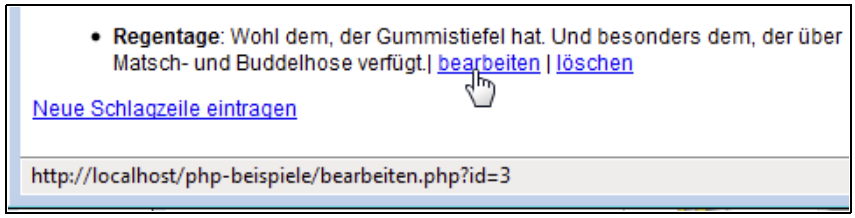


Abbildung 11.14: Wenn Sie sich im Browser die Statusleiste einblenden lassen und über die einzelnen Links fahren, sehen Sie, dass immer die entsprechende `id` *angezeigt wird.*

Wenn der Link geklickt wird, kann in der Datei `bearbeiten.php` über `$_GET["id"]` auf den übergebenen Wert zugegriffen werden und damit der richtige Datensatz ausgelesen werden.

Nach demselben Prinzip werden auch die Links zur Datei *loeschen.php* zusammengesetzt.

In Zeile 38 wird dann noch ein Link zur Datei *neu.php* eingefügt, über die sich eine neue Nachricht eintragen lässt.

Neue Nachricht verfassen

Im Skript *neu.php* wird ein Formular angezeigt, in das Sie eine neue Nachricht eintragen können.

Abbildung 11.15: Das Formular zum Eingeben einer neuen Nachricht

Listing 11.15: Eine neue Nachricht eintragen (neu.php)

```
01 <?php
02 require_once "db_daten_test.php";
03 $host = htmlspecialchars($_SERVER["HTTP_HOST"]);
04 $uri  = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
05 $extra = "anzeigen.php";
06 if (empty($_POST["titel"])) {
07 ?>
08 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
09     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
10 <html xmlns="http://www.w3.org/1999/xhtml">
11  <head>
12   <title>News eingeben</title>
13   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
14   <style type="text/css">
15   body { font-size: 80%; font-family: sans-serif; }
16   </style>
17 </head>
18 <body>
19 <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);
     ?>">
20 Titel <br />
21 <input type="text" name="titel" maxlength="25" /><br />
22 Text <br />
23 <textarea name="text" rows="5" cols="30"></textarea><br />
24 <input type="submit" />
25 </body>
26 </html>
27 <?php
28 } else {
29   if ($stmt = $mysqli->prepare("INSERT INTO aktuell (titel, text) VALUES (?, ?)"))
{
30     $titel = $_POST["titel"];
31     $text = $_POST["text"];
32     $stmt->bind_param("ss", $titel, $text);
33     $stmt->execute();
34     $stmt->close();
35     $mysqli->close();
36     header("Location: http://$host$uri/$extra");
37    }
38 }
39 ?>
```

Wenn eine neue Nachricht eingetragen wurde, soll automatisch wieder auf die *anzeigen.php*-Seite umgeleitet werden. Um hierfür die benötigte absolute URL zusammenzustellen, werden in Zeile 3–5 die entsprechenden Komponenten definiert (siehe hierzu auch Kapitel 8).

In Zeile 6 wird überprüft, ob `$_POST["titel"]` nicht gesetzt oder leer ist. Wenn dies der Fall ist, wird eine HTML-Seite inklusive Formular ausgegeben (Zeile 8 bis 26).

Falls `$_POST["titel"]` nicht `empty` ist, wurde das Formular bereits ausgefüllt und etwas beim Titel eingetragen. In diesem Fall findet die Verarbeitung statt (Zeile 28). Zuerst wird die vorbereitete Anweisung erstellt (Zeile 29), dann die Variablen definiert und gebunden (Zeile 32) und schließlich die Anweisung ausgeführt (Zeile 33). In Zeile 36 findet die Umleitung auf die Startseite statt – damit das funktioniert, darf hier im `else`-Zweig oder davor keine Ausgabe von (X)HTML-Code erfolgt sein.

Nachricht löschen

Das Löschen-Skript soll nur über die Seite *anzeigen.php* aufgerufen werden. Wenn die Nachricht gelöscht wurde, erfolgt sofort wieder eine Umleitung zu *anzeigen.php*. Deswegen wird hier kein (X)HTML-Grundgerüst benötigt.

Listing 11.16: Das Skript zum Löschen (loeschen.php)

```
01 <?php
02 require_once "db_daten_test.php";
03 $host = htmlspecialchars($_SERVER["HTTP_HOST"]);
04 $uri  = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
05 $extra = "anzeigen.php";
06 if(!isset($_GET["id"]) || !is_numeric($_GET["id"])) {
07   header("Location: http://$host$uri/$extra");
08 }
09 $id = $_GET["id"];
10 if($stmt = $mysqli->prepare("DELETE FROM aktuell WHERE id=?")) {
11   $stmt->bind_param("i",  $id);
12   $stmt->execute();
13   $stmt->close();
14   $mysqli->close();
15   header("Location: http://$host$uri/$extra");
16 }
17 ?>
```

Zu Beginn (Zeile 3–5) werden wieder die Komponenten für die absolute URL zur Umleitung definiert. In Zeile 6 wird überprüft, ob die Variable `$_GET["id"]` gesetzt ist und numerisch ist. Falls das nicht der Fall ist, bedeutet das, dass das Löschen-Skript direkt aufgerufen wurde oder die URL manipuliert wurde, und es findet direkt eine Umleitung auf die *anzeigen.php*-Seite statt (Zeile 7).

Ansonsten wird der Löschvorgang wieder über vorbereitete Statements durchgeführt (Zeile 10ff.). Am Schluss findet die Umleitung zu *anzeigen.php* statt (Zeile 15).

Bestehende Nachrichten bearbeiten

Um bestehende Nachrichten zu bearbeiten, ist etwas mehr Aufwand erforderlich. Bei der Bearbeitung gibt es zwei voneinander relativ unabhängige Vorgänge:

- Wenn das Skript über *anzeigen.php* aufgerufen wurde, wird die ID des Datensatzes übergeben, der bearbeitet werden soll. Jetzt muss Titel und Text zur entsprechenden ID über `SELECT` ausgelesen und im Formular angezeigt werden.
- Wird das Formular abgesendet, muss der `UPDATE`-Vorgang durchgeführt werden und wieder eine Umleitung zu *anzeigen.php* erfolgen.

Sehen wir uns das Skript einmal an:

Listing 11.17: Das Skript zum Bearbeiten bestehender Datensätze (bearbeiten.php)

```
<?php
require_once "db_daten_test.php";
$host = htmlspecialchars($_SERVER["HTTP_HOST"]);
$uri  = rtrim(dirname(htmlspecialchars($_SERVER["PHP_SELF"])), "/\\");
$extra = "anzeigen.php";
if (empty($_POST["titel"])) {
  if(!isset($_GET["id"]) || !is_numeric($_GET["id"])) {
     header("Location: http://$host$uri/$extra");
   }
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>News verwalten</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
  <style type="text/css">
  body { font-size: 80%; font-family: sans-serif; }
  </style>
</head>
<body>
<?php
  $id = $_GET["id"];
  if($stmt = $mysqli->prepare("SELECT id, titel, text
      FROM aktuell WHERE id=?")) {
  $stmt->bind_param("i", $id);
  $stmt->execute();
  $stmt->bind_result($id, $titel, $text);
  $stmt->fetch();
  $stmt->close();
  $mysqli->close();
  }
?>
```

```
34 <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);
     ?>">
35 Titel <br />
36 <input type="text" name="titel" value="<?php echo htmlspecialchars($titel); ?>"
     /><br />
37 Text <br />
38 <textarea name="text" rows="5" cols="30"><?php echo htmlspecialchars($text);
     ?></textarea><br />
39 <input type="hidden" name="id" value="<?php echo $id; ?>" />
40 <input type="submit" />
41 </body>
42 </html>
43 <?php
44 } else {
45   $id = (int)$_POST["id"];
46   if($stmt = $mysqli->prepare("UPDATE aktuell
47               SET titel=?, text=? WHERE id=?")) {
48     $titel = $_POST["titel"];
49     $text = $_POST["text"];
50     $stmt->bind_param("ssi", $titel, $text, $id);
51     $stmt->execute();
52     $stmt->close();
53     $mysqli->close();
54     header("Location: http://$host$uri/$extra");
55   }
56 }
57 ?>
```

Den Anfang (Zeile 1–4) kennen Sie schon. Interessant ist jetzt die Überprüfung in Zeile 6. Wenn `$_POST["titel"]` leer ist, soll das Formular mit den Inhalten angezeigt werden. Dafür ist die `id` der Nachricht notwendig, die modifiziert werden soll. Diese wird im Normalfall über die URL übergeben. Sollte diese jedoch nicht gesetzt sein, also *bearbeiten.php* ohne Parameter aufgerufen sein, oder die `id` nicht numerisch sein, findet sofort wieder eine Umleitung zu *anzeigen.php* statt (Zeile 8).

Ansonsten wird über eine vorbereitete Anweisung in Zeile 24 der gesamte Datensatz zur ID geholt und die Ergebnisse an Variablen gebunden. Da es sich nur um einen Datensatz handelt, braucht `fetch()` in Zeile 29 nicht in einer Schleife aufgerufen zu werden, sondern es genügt ein einmaliger Aufruf.

Ab Zeile 34 wird das Formular erstellt und die ausgelesenen Werte in die Formularfelder voreingetragen. In Zeile 39 wird außerdem `id` in ein verstecktes Feld eingetragen.

```
39 <input type="hidden" name="id" value="<?php echo $id; ?>" />
```

Wenn das Formular abgeschickt wurde und bei `Titel` etwas eingetragen ist, wird der `else`-Zweig ausgeführt, der in Zeile 44 beginnt. Zuerst wird eine vorbereitete Anwei-

sung für den UPDATE-Vorgang erstellt (Zeile 46ff.). Damit der richtige Datensatz verändert wird, ist die id notwendig – und diese wurde ja über das versteckte Feld übertragen und steht damit in $_POST["id"].

Tipp

Beim ersten Aufruf von *bearbeiten.php* über die *anzeigen.php*-Seite hängt am Dateinamen die id des Datensatzes (*beabeiten.php?id=2*). Wenn das Formular hingegen abgesendet wird, wird es an *bearbeiten.php* gesendet – ohne Parameter. Dafür, dass die id jetzt ausgelesen werden kann, wurde sie im Formular in ein verstecktes Feld eingetragen.

Am Schluss wird die vorbereitete Anweisung ausgeführt (Zeile 51) und es findet wieder eine Umleitung zu *anzeigen.php* (Zeile 54) statt.

11.6 Alternativen: MySQLi-Schnittstelle prozedural und MySQL-Schnittstelle

Bisher wurde durchgängig die MySQLi-Schnittstelle objektorientiert genutzt, Sie können sie aber auch prozedural verwenden. Hierzu ein Beispiel, in dem alle Datensätze der Tabelle pflanzen ausgelesen und als Tabelle angezeigt werden:

Listing 11.18: Daten als Tabelle ausgeben (db_ausgabe_prozedural.php)

```
01 $db = mysqli_connect("localhost", "root", "geheim", "garten");
02 if (!$db) {
03   echo "Fehler bei der Verbindung: " . mysqli_connect_error();
04   exit();
05 }
06 $ergebnis = mysqli_query($db, "SELECT name, beschreibung, preis FROM pflanzen;");
07 echo "<table border='1'>\n";
08 while($zeile = mysqli_fetch_array($ergebnis)) {
09   echo "<tr><td>" . htmlspecialchars($zeile["name"]) . "</td>"
10        . "<td>" . htmlspecialchars($zeile["beschreibung"]) . "</td>"
11        . "<td>" . htmlspecialchars($zeile["preis"]) . "</td>"
12        . "</tr>\n";
13 }
14 echo "</table>";
15 mysqli_close($db);
```

Die Verbindung wird in Zeile 1 über mysqli_connect() hergestellt, das einen Handle für die Verbindung zurückgibt. Dieser muss in Zeile 6 zur Durchführung der Abfrage als erster Parameter an mysqli_query() übergeben werden. mysqli_query() gibt eine Ergebniskennung zurück, die dann in Zeile 8 an mysli_fetch_array() übergeben wird, um die einzelnen Datensätze auszulesen.

Auch der Einsatz der Schnittstelle MySQL anstelle von MySQLi funktioniert ähnlich, weicht aber im Detail ab. Ein Beispiel, in dem die Inhalte der Tabelle `pflanzen` ausgelesen und in einer (X)HTML-Tabelle ausgegeben werden, finden Sie bei den Buchbeispielen unter dem Namen *db_ausgabe_mysql.php* als Listing.

Datenbanken sind ungeheuer mächtig und ermöglichen eine komfortable Verwaltung von Daten aller Art. Aber PHP kann selbstverständlich auch mit Textdateien arbeiten. Mehr dazu im nächsten Kapitel.

12 Dateien lesen und schreiben, Verarbeitung von XML und von Phar-Archiven

PHP stellt Ihnen viele praktische Funktionen für die Arbeit mit Dateien zur Verfügung. Sie können auf die Inhalte von lokalen Dateien genauso zugreifen wie auch Dateien auf entfernten Webservern auslesen. Eine eigene Schnittstelle macht das Auslesen von XML-Dateien zum Kinderspiel – Sie sehen in diesem Kapitel, wie leicht man so die Newsfeeds von anderen Seiten ausgeben kann. Schließlich erfahren Sie, was es mit den Phar-Archiven auf sich hat.

12.1 Wichtige Basis: Dateirechte

Um auf Dateien zuzugreifen, müssen Sie die entsprechenden Rechte besitzen. Auf Linux/Unix-Systemen gibt es ein ausgeklügeltes Rechtesystem. Da die meisten Server im Internet unter Linux laufen, ist es wichtig, die Grundlagen zu kennen.

Unterschieden werden drei verschiedene Rechte:

- Leserechte (abgekürzt als `r`): Um eine Datei zu lesen, benötigen Sie Leserechte für diese Datei.
- Schreibrecht (`w`): Um eine Datei beschreiben zu können, brauchen Sie Schreibrechte.
- Recht zum Ausführen (`x`): Damit können Sie beispielsweise eine Datei als Programm starten.

Hinweis

Diese verschiedenen Typen von Rechten gelten auch für Ordner: Allerdings brauchen Sie sinnvollerweise für Ordner, um die Dateien genauer anschauen zu können, Schreib- und *Ausführrechte*.

Zudem werden drei Gruppen von Nutzern unterschieden:

- Der Besitzer einer Datei/eines Ordners
- Die Gruppe, zu der der Besitzer gehört
- Andere (der Rest der Welt)

Wie die Rechte bei einzelnen Dateien oder Ordnern vergeben sind, wird durch eine *zehnstellige* Anzeige dargestellt. Ein Beispiel, wie die Dateirechte im FTP-Programm Filezilla dargestellt werden, zeigt Abbildung 12.1.

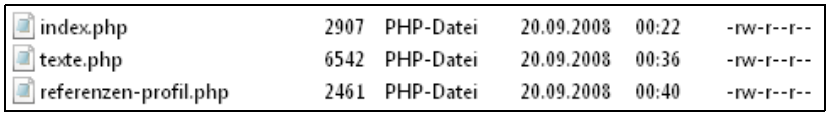


Abbildung 12.1: Dateirechte in Filezilla: Die Angaben sehen Sie in der letzten Spalte.

Filezilla zeigt bei den einzelnen Dateien als Berechtigungen `-rw-r--r--` an. Dies lässt sich in vier Bestandteile zerlegen:

- `-`: An erster Stelle steht `-`, das kennzeichnet hier, dass es sich um eine Datei handelt. Bei einem Ordner stünde stattdessen `d` (*directory*).

Darauf folgen die jeweils dreistelligen Angaben zu den Rechten der Benutzer:

- `rw-`. Die erste Dreiergruppe bezeichnet die Rechte des Eigentümers. `rw-` in der Abbildung bedeutet beispielsweise, dass der Besitzer Lese- und Schreibrechte, aber keine Ausführrechte besitzt.
- `r--` Das darauf folgende `r--` charakterisiert die Rechte der Gruppe.
- `r--` Die zuletzt folgende Dreiergruppe beschreibt die Rechte von anderen an dieser Datei, beispielsweise ebenfalls `r--`.

Diese Rechte können Sie über Ihr FTP-Programm ändern. Bei Filezilla erledigen Sie das, indem Sie auf die Datei/das Verzeichnis mit der rechten Maustaste klicken und DATEIATTRIBUTE wählen. Dann lassen sich die Rechte durch einen Klick in die entsprechenden Kästchen vergeben.

Sie sehen bei Filezilla in Abbildung 12.2 auch einen numerischen Wert, nämlich 666. Die Rechte können nämlich auch auf eine andere Art angegeben werden. Dabei steht:

- 4 für das Leserecht
- 2 für Schreibrecht
- 1 für das Recht, eine Datei auszuführen

Diese Werte werden addiert. Wiederum stehen die Rechte für den Benutzer vor den Rechten für die Gruppe und den Rest der Welt. 755 bedeutet beispielsweise: Alle Rechte (Lesen, Schreiben, Ausführen = 4+2+1) für den Besitzer, Leserecht und Ausführrecht für die Gruppe (4+1) und den Rest der Welt und wäre ein typischer Wert, den Sie brauchen, wenn Sie möchten, dass PHP in ein Verzeichnis schreiben kann.

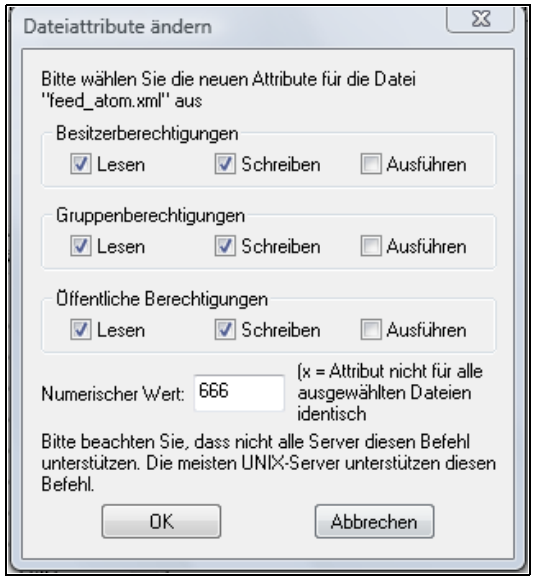


Abbildung 12.2: Die Rechte in Filezilla

Tipp

Falls Sie die Dateirechte per PHP setzen möchten, so geht das auch über den Befehl `chmod()`. Dieser erwartet als ersten Parameter die Datei und als zweiten die Rechte in oktaler Form, es muss hier eine 0 vorangestellt werden.

```
chmod("beispieldatei.txt", 0666);
```

12.2 Schnell zum gewünschten Ziel über file_get_contents() und file_put_contents()

Es gibt verschiedene Methoden, um per PHP auf Dateien zuzugreifen oder auch in Dateien zu schreiben. Sehr komfortabel und mit wenig Aufwand lassen sich die Funktionen `file_get_contents()` und `file_put_contents()` einsetzen.

12.2.1 Inhalte schnell auslesen

`file_get_contents()` liest den Inhalt einer Datei ein und speichert ihn in einem String. Für das folgende Beispiel benötigen Sie eine kleine Textdatei mit Namen *beispieldatei.txt*, die sich im selben Verzeichnis wie das PHP-Skript befinden muss. Sie sieht so aus:

Listing 12.1: Die Textdatei beispieldatei.txt

```
Eine Beispieldatei mit mehreren Zeilen Text.
Das war die erste und hier ist die zweite.
Eine dritte folgt hier.
```

Diese lässt sich dann per PHP auslesen (Listing 12.2).

Listing 12.2: Den Text aus einer Datei auf einmal auslesen (text_auslesen.php)

```
$datei = "beispieldatei.txt";
$inhalt = file_get_contents($datei);
echo strtoupper($inhalt);
```

Der Inhalt der Datei *beispieldatei.txt* wird in der Variablen `$inhalt` gespeichert und in Großbuchstaben ausgegeben.

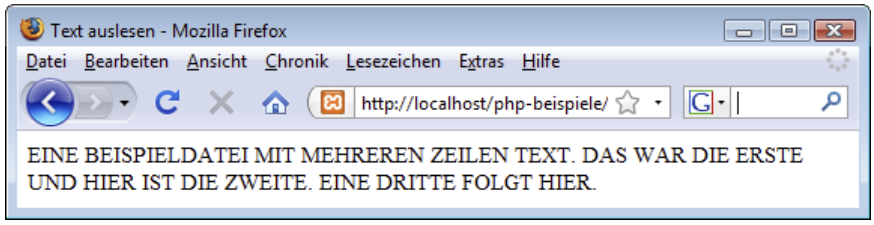


Abbildung 12.3: Der Inhalt der Datei in Großbuchstaben

Achtung

Vorsicht ist geboten, wenn Sie Benutzern erlauben, über ein Formularfeld anzugeben, welche Datei geöffnet werden soll. Dabei besteht die Gefahr, dass ein Benutzer einen manipulierten Dateinamen angibt und sich dadurch eine Datei mit nicht für ihn gedachten Informationen anzeigen lässt. Deswegen sollten Sie in diesem Fall immer die eingegebene Datei gegen eine Liste von erlaubten Dateien prüfen oder zumindest verbotene Zeichen wie . . / entfernen.

In diesem Beispiel wird jedoch nicht überprüft, ob die Datei überhaupt eingelesen werden kann. Falls Sie sich beim Dateinamen verschrieben haben, erhalten Sie eine hässliche Fehlermeldung. Deswegen sollten Sie vor der Ausgabe eine Prüfung durchführen. Verwenden können Sie hierfür, dass `file_get_contents()` im Fehlerfall `false` zurückgibt. Ein erster Ansatz könnte folgendermaßen aussehen:

```
$datei="datei.txt";
$inhalt = @file_get_contents($datei);
if ($inhalt == false) {
  echo "Hat nicht geklappt";
}
```

Sie sehen, hier werden erst einmal über das @-Zeichen vor `file_get_contents()` eventuelle Warnungen unterdrückt. Dann wird der Rückgabewert überprüft.

Allerdings gibt es noch einen Schönheitsfehler: Falls in der eingelesenen Datei `0` steht oder die Datei leer ist, würde ebenfalls »Hat nicht geklappt« ausgegeben: Durch das automatische Typecasting verwandelt PHP `0` in `false`. Deswegen brauchen Sie hier den Identitätsoperator ===, der bei der Prüfung auch den Typ berücksichtigt:

Listing 12.3: Integrierte Überprüfung (text_auslesen_mit_pruefung.php)

```
$datei="datei.txt";
$inhalt = @file_get_contents($datei);
if ($inhalt === false) {
  echo "Hat nicht geklappt";
}
else {
  echo "Inhalt ist: $inhalt";
}
```

Im Beispiel wird jetzt die *datei.txt* eingelesen, die nur die `0` enthält. Es erscheint die richtige Meldung.

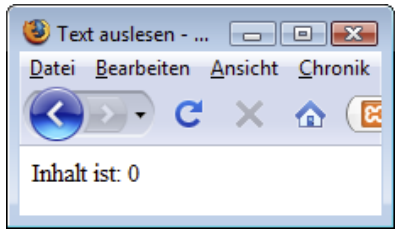


Abbildung 12.4: Auch wenn in der Datei nur eine 0 steht, klappt das Auslesen.

Bisher hat sich die auszulesende Datei immer im selben Verzeichnis befunden, in dem auch das PHP-Skript gespeichert ist. Sollte das nicht der Fall sein, müssen Sie die Pfade entsprechend anpassen.

Tipp

Beachten Sie, dass Sie bei Windows-Systemen den sonst unter Windows nicht üblichen / als Trennzeichen verwenden können. Der Vorteil: Diese Schreibung funktioniert unabhängig vom eingesetzten Betriebssystem.

Unter Windows könnten Sie allerdings auch den Backslash benutzen, müssen diesen aber maskieren:

```
$datei="unterverzeichnis\\beispieldatei.txt";
```

Das klappt dann aber nur unter Windows.

Hinweis

`file_get_contents()` können Sie benutzen, um die Inhalte von anderen Webseiten einzulesen. Dies klappt allerdings nur, wenn die Konfigurationseinstellung `allow_url_fopen` auf `On` steht. Diese Einstellung nehmen Sie in der *php.ini* vor. Ein Beispiel, wie sich `file_get_contents()()` zum Zugriff auf entfernte Dateien nutzen lässt, finden Sie unter dem Namen *externe_dateien_auslesen.php* bei den Listings dieses Buchs.

12.2.2 In Dateien schreiben

Parallel zu `file_get_contents()` gibt es `file_put_contents()`, um in eine Datei zu schreiben. Diese Funktion erwartet zwei Parameter: zuerst den Namen der Datei, in die geschrieben werden soll, und dann den zu schreibenden Inhalt. Im folgenden Listing wird ein einfacher Seitenaufrufzähler realisiert.

Listing 12.4: Mit file_put_contents() schreibt man in eine Datei (text_schreiben.php).

```
$datei = "beschreibbar/zaehler.txt";
$anzahl = @file_get_contents($datei);
if ($anzahl !== false) {
  $anzahl++;
  @file_put_contents($datei, $anzahl);
  echo "Dies ist der $anzahl. Aufruf";
}
```

Voraussetzung für diesen Zähler ist, dass die Datei *zaehler.txt* im angegebenen Ordner existiert, beschreibbar ist und dass darin ein Anfangswert steht. Über `file_get_contents()` wird der aktuelle Wert ausgelesen, dann wird er um eins hochgezählt und wieder erneut in die Datei geschrieben. Wenn Sie die Seite mehrmals aufrufen, sehen Sie, dass der Zähler hochgezählt wird.

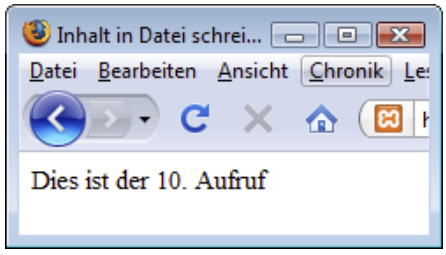


Abbildung 12.5: Die Anzahl der Aufrufe wird gezählt.

Tipp

Dies ist ein sehr einfacher Zähler, weil er auch bei einem Reload hochzählt. Wenn Sie wollen, dass ein Reload nicht gezählt wird, sondern nur Besuche von verschiedenen Besuchern oder Besuche zwischen denen eine gewisse Zeitspanne liegt, so müssen Sie mehr Aufwand betreiben.

Prinzipiell funktioniert `file_put_contents()` ebenso wie `file_get_contents()` auch bei entfernten Dateien. Meist aber besitzen Sie bei entfernten Webservern keine Schreibrechte – außer per FTP.

Tipp

Mit PHP 5.3 wurde das phar-Dateiformat für Dateiarchive eingeführt. In Abschnitt 12.5.2 finden Sie ein Beispiel, wie Sie mit `file_get_contents()` auf solche Dateien zugreifen.

12.3 Schritt für Schritt mit fopen() & Co.

Die beiden vorgestellten Funktionen sind wunderbar, da alles »automatisch« geschieht. Mehr Kontrolle haben Sie allerdings, wenn Sie die Schritte einzeln durchführen – damit können Sie beispielsweise Dateien auch zeilenweise auslesen und bearbeiten.

12.3.1 Datei öffnen in verschiedenen Modi

Zuerst müssen Sie hierfür die Datei mit der Funktion `fopen()` öffnen. Schreiben Sie bei `fopen()` als ersten Parameter den Dateinamen. Als zweiten Parameter geben Sie an, *wozu* Sie die Datei öffnen möchten. Dabei wird unterschieden, ob Sie die Datei zum Lesen (`r`) oder zum Schreiben (`w`, `x`, `a`) öffnen wollen. Außerdem wird bei bestimmten Modi ein möglicherweise bestehender Inhalt überschrieben (`w`) oder nicht (`a`). Zusätzlich kann noch unterschieden werden, was passieren soll, wenn die Datei noch nicht existiert. Die möglichen Modusangaben zeigt Tabelle 12.1.

Modusangabe	Funktion
`r`	Lesen
`r+`	Lesen und Schreiben. Der Dateizeiger wird am Anfang platziert und von dort aus eventuell vorhandener Text überschrieben.
`w`	Schreiben. Falls die Datei bereits einen Inhalt hat, wird dieser gelöscht. Wenn die Datei nicht existiert, wird versucht, sie anzulegen.
`w+`	Lesen und Schreiben. Falls die Datei bereits einen Inhalt hat, wird dieser gelöscht. Wenn die Datei nicht existiert, wird versucht, sie anzulegen.
`x`	Erstellt eine neue Datei, um in sie zu schreiben. Falls die Datei schon existiert, wird `false` zurückgegeben und eine Warnung ausgegeben.
`x+`	Erstellt eine neue Datei, um in sie zu schreiben. Falls die Datei schon existiert, wird `false` zurückgegeben und eine Warnung ausgegeben.
`a`	Beim Schreiben in die Datei werden die Daten angehängt, das bedeutet, bestehende nicht überschrieben. Falls die Datei noch nicht existiert, wird versucht, sie anzulegen.
`a+`	Lesen und Schreiben. Beim Schreiben in die Datei werden die Daten angehängt, das bedeutet, bestehende nicht überschrieben. Falls die Datei noch nicht existiert, wird versucht, sie anzulegen.
`b`	Binary-Modus. Dies ist nur bei Windows-Systemen relevant. Wird mit den anderen Modiangaben kombiniert.
`t`	Textmodus, also das Gegenteil zu `b`. Nur sinnvoll bei Windows. Wird mit den anderen Modiangaben kombiniert.

Tabelle 12.1: Modusangaben bei fopen()

Hinweis

Noch eine Anmerkung zu `t` und `b`. Diese sind nur bei Windows relevant. Der Hintergrund hierfür ist, dass verschiedene Betriebssysteme unterschiedliche Zeichen für die Zeilenenden verwenden. Windows benutzt `\r\n`, Unix/Linux hingegen `\n` und Mac OSX `\r`. Wenn Sie bei Windows den Textmodus (`t`) wählen, dann wird `\n` durch `\r\n` bei der Arbeit mit der Datei ersetzt. Das PHP-Manual empfiehlt, `t` zu verwenden, wenn Sie als Zeilenbegrenzer `\n` verwenden und sicherstellen möchten, dass die Datei korrekt mit einer Anwendung wie Notepad dargestellt wird. In allen anderen Fällen sollten Sie `b` benutzen.

So öffnen Sie eine Datei zum Lesen:

```
$dh = fopen("beispieldatei.txt", "rb");
```

`fopen()` gibt Ihnen dann einen Dateihandle zurück, den Sie weiteren Funktionen zur Arbeit mit Dateien übergeben, beispielsweise `fgets()` und `feof()`.

12.3.2 Zeilenweise auslesen

Um den Inhalt einer Datei zeilenweise auszulesen, verwenden Sie `fgets()`, das Ihnen den Inhalt einer einzelnen Zeile zurückgibt. Dann wird der Dateizeiger weiterbewegt, so dass beim nächsten Aufruf von `fgets()` die nächste Zeile gelesen wird. Am besten setzen Sie deswegen `fgets()` in einer Schleife ein: In der Bedingung der Schleife überprüfen Sie mit `feof()`, ob das Ende der Datei erreicht ist.

Listing 12.5: Mit fgets() lesen Sie zeilenweise aus einer Datei (text_zeilenweise.php).

```
$datei="beispieldatei.txt";
$dh = fopen($datei, "r");
echo "<ul>\n";
while (!feof($dh)) {
  $zeile = fgets($dh);
  echo "<li>" . htmlspecialchars($zeile) . "</li>\n";
}
echo "</ul>\n";
fclose($dh);
```

Im Beispiel wird eine Datei zum Lesen geöffnet. Der Inhalt wird mit `fgets()` zeilenweise ausgelesen und in eine ungeordnete Liste geschrieben. Sicherheitshalber werden mögliche HTML-Sonderzeichen mit `htmlspecialchars()` geschützt. Am Ende wird der Dateihandle mit `fclose()` wieder geschlossen.

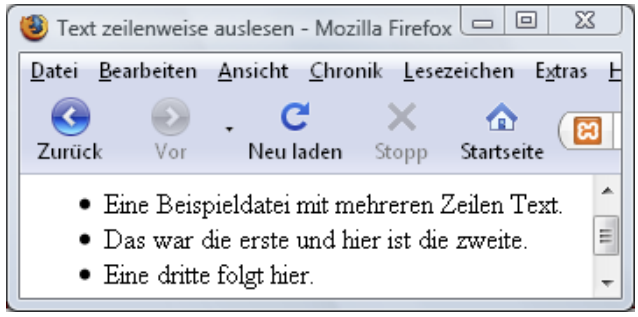


Abbildung 12.6: Der Inhalt der Datei wird als ungeordnete Liste ausgegeben.

Zum Auslesen von Daten können Sie neben `fgets()` auch `fread()` nehmen. Es erwartet als ersten Parameter ein Dateihandle und als zweiten die Anzahl an Bytes, die gelesen werden soll.

```
$inhalt = fread($dh, 150);
```

12.3.3 In Dateien schreiben

`fwrite()` dient zum Schreiben in Dateien. Geben Sie als ersten Parameter den Dateihandle und als zweiten den zu schreibenden Inhalt an:

Listing 12.6: Zum Schreiben von Inhalten können Sie fwrite() benutzen (text_schreiben_fwrite.php).

```
$inhalte = array("Riesenrad", "Achterbahn", "Schießstand");
$datei ="beschreibbar/beispiel.txt";
$dh = fopen($datei, "w");
foreach ($inhalte as $inhalt) {
  fwrite($dh, "$inhalt ");
}
fclose($dh);
```

Im Beispiel wird eine Datei mit `fopen()` im Modus `w` geöffnet: Das heißt, sie wird zum Schreiben geöffnet, und falls sie nicht existiert, wird versucht, sie anzulegen. Dann wird der Inhalt eines vorher definierten Arrays in die Datei geschrieben. Das PHP-Skript erzeugt keine im Browserfenster sichtbare Ausgabe. Wenn Sie aber danach in die Datei *beispiel.txt* im Ordner *beschreibbar* schauen, sollte der Inhalt des Arrays dort hineingeschrieben sein.

12.3.4 Prüfungen durchführen

Das Beispiel funktioniert natürlich nur, wenn das Verzeichnis auch beschreibbar ist. Ob das der Fall ist, können Sie mit `is_writable()` feststellen. Diese Funktion gibt `true` zurück, wenn das Verzeichnis beschreibbar ist, ansonsten `false`.

Parallel dazu überprüft `is_readable()`, ob eine Datei lesbar ist. Ob eine Datei existiert, prüft hingegen `file_exists()` und mit `is_file()` können Sie feststellen, ob etwas überhaupt eine Datei ist.

Diese Funktionen können bei einer Fehlerbehandlung mit Exceptions (Genaueres zur Exception-Klasse finden Sie in Kapitel 9) eingesetzt werden, wie das folgende Listing zeigt: Hier wird eine Klasse `Dateilesen` definiert: Diese definiert Methoden zum Öffnen, Lesen und Schließen einer Datei. Bei den einzelnen Schritten werden Überprüfungen durchgeführt und bei Problemen Ausnahmen ausgelöst:

Listing 12.7: Die Exception-Klasse für mögliche Fehler beim Dateilesen einsetzen (datei_pruefen_exception.php)

```
01 class Dateilesen
02 {
03   protected $datei;
04   protected $dh;
05   public function __construct($datei)
```

```
06   {
07    $this->datei = $datei;
08   }
09   public function oeffnen()
10   {
11     if (!file_exists($this->datei)) {
12       throw new Exception("Datei gibt's nicht");
13     }
14     if (!is_readable($this->datei)) {
15       throw new Exception("nicht lesbar");
16     }
17     $this->dh = @fopen($this->datei, "r");
18     if ($this->dh == false) {
19       throw new Exception("ging nicht");
20     }
21   }
22   public function lesen()
23   {
24     $zeile = fgets($this->dh, 1024);
25     if ($zeile === false) {
26       throw new Exception("Lesen gescheitert");
27     }
28     return htmlspecialchars($zeile);
29   }
30   public function schliessen()
31   {
32     fclose($this->dh);
33   }
34 }
35 try {
36   $meinedatei = new Dateilesen("beschreibbar/zaehler.txt");
37   $meinedatei->oeffnen();
38   echo $meinedatei->lesen();
39   $meinedatei->schliessen();
40 } catch (Exception $e) {
41   echo "Folgender Fehler ist aufgetreten: "
42          . $e->getMessage();
43 }
```

In der Konstruktormethode in Zeile 5 wird eine Datei entgegengenommen und der Eigenschaft `$datei` zugewiesen. In der `oeffnen()`-Methode in Zeile 9 finden mehrere Überprüfungen statt, so beispielsweise, ob die Datei existiert (Zeile 11) oder ob sie überhaupt lesbar ist (Zeile 14). Ansonsten wird die Datei in Zeile 17 geöffnet.

Die Methode `lesen()` in Zeile 22 liest eine Zeile und gibt sie mit `htmlspecialchars()` behandelt zurück (Zeile 28). Wenn das Lesen nicht funktioniert, wird wieder eine Exception ausgelöst. Schließlich gibt es noch die `schliessen()`-Methode ab Zeile 30.

Der restliche Code teilt sich in einen `try`- und einen `catch`-Zweig. Im `try`-Zweig ab Zeile 35 wird ein neues Objekt der `Dateilesen`-Klasse erstellt und versucht, die angegebene Datei zu öffnen und zu lesen. Falls es Fehler gibt, werden diese durch den `catch`-Zweig ab Zeile 40 abgefangen. Über `$e->getMessage()` wird die Fehlermeldung ausgegeben.

Hinweis

Sie haben jetzt Beispiele gesehen, wie Sie mit `fopen()` eine Datei öffnen. Möchten Sie hingegen Dateien nicht öffnen, sondern beispielsweise nur die Dateinamen auslesen, können Sie mit `opendir()` ein Verzeichnis öffnen und mit `readdir()` seinen Inhalt auslesen. Ein Beispiel dazu finden Sie in Kapitel 13, wenn es darum geht, zu allen JPEG-Bildern eines Verzeichnisses kleine Vorschaubilder zu erstellen.

12.4 XML-Dateien auslesen

Nun geht es um eine besondere Art von Dateien, nämlich die XML-Dateien.

XML steht für eXtensible Markup Language und ist eigentlich eine Metasprache, die festlegt, wie Markup-Sprachen zu definieren sind. Bei konkreten XML-Dokumenten kann man selbst entscheiden, wie die Elemente heißen sollen. XML wird im Allgemeinen für den Datenaustausch benutzt, neben XHTML, das eine XML-Anwendung ist, gibt es viele weitere wie beispielsweise DocBook für Dokumentationen oder RSS/Atom für Newsfeeds. Bevor wir dazu kommen, wie Sie auf Newsfeeds von anderen Seiten zugreifen, sehen wir uns erst einmal an, wie Sie mit PHP XML-Dateien verarbeiten können.

12.4.1 Zugriff auf XML-Dateien – Grundlagen

Das Folgende ist ein Beispiel für ein kleines XML-Dokument:

Listing 12.8: Ein Beispiel für ein XML-Dokument (xml_beispiel.xml)

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <person id="nr1">
03   <name>
04     <vorname>Werner</vorname>
05     <nachname>Weberknecht</nachname>
06   </name>
07   <adresse art="privat">
08     <plz>12345</plz>
09     <ort>Am Hügel</ort>
10   </adresse>
11 </person>
```

XML-Dokumente bestehen immer aus einem Wurzelelement, das alles andere umfasst. In diesem Beispiel ist es `person`. Darin verschachtelt können weitere Elemente stehen, die immer aus Start- und End-Tag bestehen. Elemente können über Attribute näher gekennzeichnet werden – im Beispiel `art="privat"` in Zeile 7.

Hinweis

Alle XML-Formate folgen den XML-Regeln. Diese besagen beispielsweise, dass zu jedem Start-Tag auch ein End-Tag gehört oder dass Attributwerte in Anführungszeichen stehen müssen. Die einzelnen XML-Dokumente unterscheiden sich nun darin, wie die Elemente heißen, wie sie verschachtelt sind und wo welche Attribute vorgesehen sind. Ein XML-Dokument, das diesen Regeln folgt, wird als wohlgeformt bezeichnet. Nur wohlgeformte Dokumente sind XML-Dokumente.

PHP bietet seit Version 5 eine einfache Möglichkeit, auf die Inhalte von XML-Dokumenten zuzugreifen und auch XML-Dokumente selbst zu erstellen. Die magische Erweiterung heißt SimpleXML und wird ihrem Namen gerecht. Dabei wird das XML-Dokument in ein Objekt konvertiert. XML-Knoten sind dabei Eigenschaften, Attribute können über Array-Indizes angesprochen werden.

Sehen wir uns ein Beispiel an: Über `simplexml_load_file()` wird eine XML-Datei geladen. Zurückgegeben wird ein Objekt, über das `print_r()` einen Überblick verschafft:

Listing 12.9: XML-Daten laden (xml_load_file.php)

```
$xmldatei = "xml_beispiel.xml";
$xml = simplexml_load_file($xmldatei);
echo "<pre>";
print_r($xml);
echo "</pre>";
```

Abbildung 12.7 zeigt die Grundstruktur: Ein SimpleXML-Element-Objekt, wobei die Unterelemente selbst wieder, sofern sie weitere Unterelemente haben, SimpleXML-Element-Objekte sind.

Hinweis

Einen Fall von verschachtelten Objekten finden Sie auch in Kapitel 9.

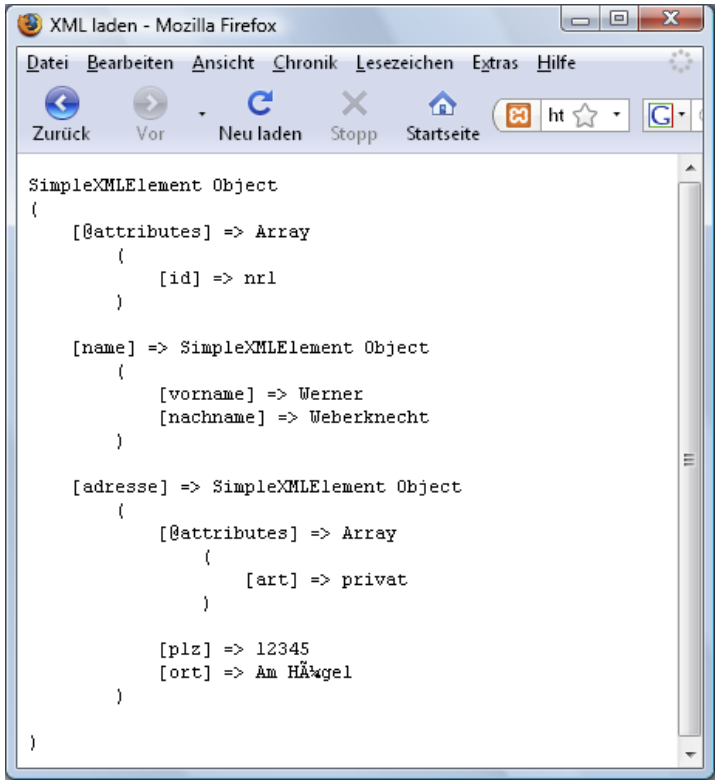


Abbildung 12.7: So sieht die Grundstruktur des XML-Objekts aus – um das missglückte Sonderzeichen beim Ort kümmern wir uns gleich noch.

Schauen wir uns nun an, wie man auf die einzelnen Inhalte zugreifen kann: Den Vornamen geben Sie so aus:

```
echo $xml->name->vorname;
```

Auf Attribute können Sie zugreifen, indem Sie das Element als Array behandeln und den Namen des Attributs als Index notieren. Das Wurzelelement `person` brauchen Sie nicht mehr anzugeben.

```
echo $xml["id"];
```

Ebenso können Sie auf das Attribut des Elements `adresse` zugreifen:

```
echo "Die Adresse ist " . $xml->adresse["art"];
```

Noch eine kleine Modifikation: Das XML-Dokument ist im Zeichensatz UTF-8 erstellt, was der Standardzeichensatz für XML-Dokumente ist. Im Dokument, in dem wir auf den Inhalt zugreifen, verwenden wir hingegen ISO-8859-1 als Zeichensatz.

Deswegen gibt es Probleme bei Sonderzeichen, wie Sie auch in Abbildung 12.7 sehen. Um das zu verhindern, müssen Sie die Texte vor der Ausgabe mit `utf8_decode()` behandeln. Das folgende Listing zeigt dies im Zusammenhang.

Listing 12.10: Einzelne Elemente gezielt ansprechen (xml_daten_ausgeben.php)

```
01 $xmldatei = "xml_beispiel.xml";
02 $xml = simplexml_load_file($xmldatei);
03 echo "<p>"
04      . utf8_decode($xml->name->vorname)
05      . " "
06      . utf8_decode($xml->name->nachname)
07      . " wohnt in "
08      . utf8_decode($xml->adresse->plz)
09      . " "
10      . utf8_decode($xml->adresse->ort)
11      . ". <br /> Diese Adresse ist "
12      . utf8_decode($xml->adresse["art"])
13      . ".</p>";
```

Abbildung 12.8: Auf einzelne Elemente zugreifen – und jetzt klappt es auch mit den Sonderzeichen.

Häufig haben Sie bei XML-Dokumenten mehrere gleichnamige Elemente. Diese können Sie dann mit einer `foreach`-Schleife durchlaufen oder die einzelnen über ihren numerischen Index ansprechen. Um dies zu demonstrieren, wird die XML-Datei modifiziert. Sie soll nun zwei Adresselemente enthalten:

Listing 12.11: Dieses Mal gibt es zwei adresse-Elemente (xml_beispiel_2.xml).

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <person id="nr1">
03   <name>
04     <vorname>Werner</vorname>
05     <nachname>Weberknecht</nachname>
06   </name>
07   <adresse art="privat">
```

```
08     <plz>12345</plz>
09     <ort>Am Hügel</ort>
10   </adresse>
11   <adresse art="beruflich">
12     <plz>12345</plz>
13     <ort>Stadt</ort>
14   </adresse>
15 </person>
```

Listing 12.12: Mehrere gleichnamige Elemente ausgeben lassen (xml_foreach.php)

```
01 $xmldatei= "xml_beispiel_2.xml";
02 $xml = simplexml_load_file($xmldatei);
03 /* alle durchlaufen */
04 foreach($xml->adresse as $adr) {
05   echo utf8_decode($adr["art"])
06        . ": "
07        . utf8_decode($adr->plz)
08        . " "
09        . utf8_decode($adr->ort )
10        . "<br />\n";
11 }
12 /* auf die erste Adresse zugreifen */
13 echo "Die erste Adresse ist ";
14 echo utf8_decode($xml->adresse[0]->plz)
15      . " "
16      . utf8_decode($xml->adresse[0]->ort);
```

In Listing 12.12 sind es nun zwei `adresse`-Elemente in der XML-Datei. Um alle zu durchlaufen, verwenden Sie eine `foreach`-Schleife (Zeile 4), einzelne Elemente sprechen Sie über ihren numerischen Index an, wie die Zeilen 14 und 16 zeigen.

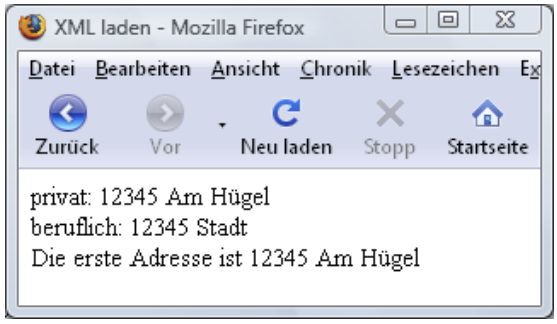


Abbildung 12.9: Ausgabe von mehreren gleichnamigen Elementen

Tipp

Bisher haben wir die XML-Daten immer aus einer Datei geladen, Sie können die XML-Daten jedoch auch als String übergeben. Dann brauchen Sie die Funktion `simplexml_load_string()`:

Listing 12.13: XML kann auch direkt aus einer Datei geladen werden (xml_aus_string.php).

```
$xmldatei = "<gruss>Hallo</gruss>";
$xml = simplexml_load_string($xmldatei);
print_r($xml);
```

12.4.2 Auf Newsfeeds zugreifen

Sie können der Funktion `simplexml_load_file()` auch eine URL übergeben und damit auf XML-Daten auf einem entfernten Server zugreifen. Dies soll am Beispiel von Newsfeeds gezeigt werden.

Tipp

Newsfeeds sind aktuelle Nachrichten, die Webseiten im XML-Format bereitstellen. Diese können Benutzer mit einem Newsfeedsreader lesen oder sie können – wie gleich gezeigt – über ein Skript drauf zugreifen. Genauere Informationen zu Newsfeeds für Anwender finden Sie beispielsweise unter *http://www.bezreg-koeln.nrw.de/brk_internet/hilfe/rss/*.

Als Beispiel sollen die Schlagzeilen der News von *http://php.net/* als Liste ausgegeben werden. Die einzelnen Titel sollen verlinkt sein und direkt auf den vollständigen Eintrag führen (Abbildung 12.1).

Zuerst einmal brauchen Sie die URL des Newsfeeds. Diese können Sie herausfinden, wenn Sie auf der Webseite von *http://www.php.net/* auf das Newsfeed-Icon klicken.

Damit wissen Sie die URL, jetzt müssen Sie sich die Struktur der XML-Datei ansehen und insbesondere, wo die gewünschten Informationen – Titel und Link – stehen. Die Struktur erhalten Sie über folgenden Vierzeiler:

Listing 12.14: So lassen Sie sich schnell die Struktur eines Newsfeeds ausgeben (atom_php_net_struktur.php).

```
$xml = simplexml_load_file("http://www.php.net/feed.atom");
echo "<pre>";
print_r($xml);
echo "</pre>";
```

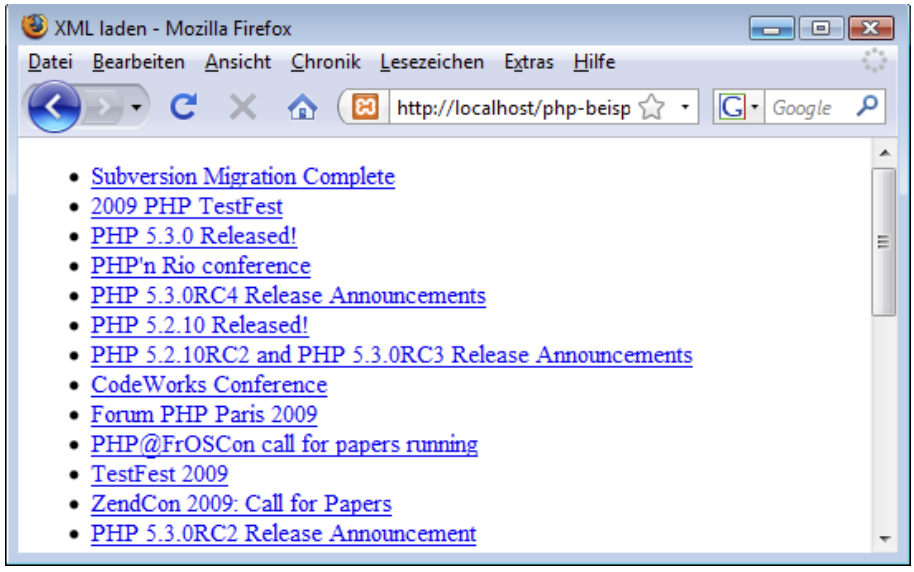


Abbildung 12.10: Newsfeed von php.net ausgeben lassen

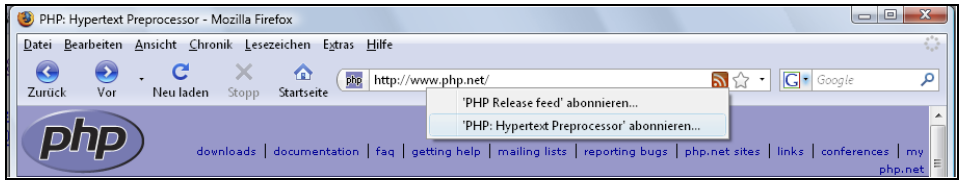


Abbildung 12.11: Die URL des Newsfeeds von php.net ermitteln

Sie sehen, im Dokument gibt es mehrere `entry`-Elemente, die die einzelnen Nachrichten enthalten. Die Informationen, die uns interessieren, sind Unterelemente von `entry`:

- `title` beinhaltet den Titel der Nachricht. Auf den Titel der ersten Nachricht können Sie also per `$xml->entry[0]->title` zugreifen.
- Im `href`-Attribut des `link`-Elements steht die URL des vollständigen Beitrags. Die URL der ersten Nachricht erreichen Sie über `$xml->entry[0]->link["href"]`.

Um die Titel und die Links aller `entry`-Elemente auszulesen, werden die `entry`-Elemente in einer `foreach`-Schleife durchlaufen. Sicherheitshalber werden Titel und URL vor der Ausgabe mit `htmlspecialchars()` und `utf8_encode()` behandelt. Da an sich `$xml->entry[0]->title` ein Objekt ist und kein String, wird es vorher noch per `(string)` in einen String umgewandelt.

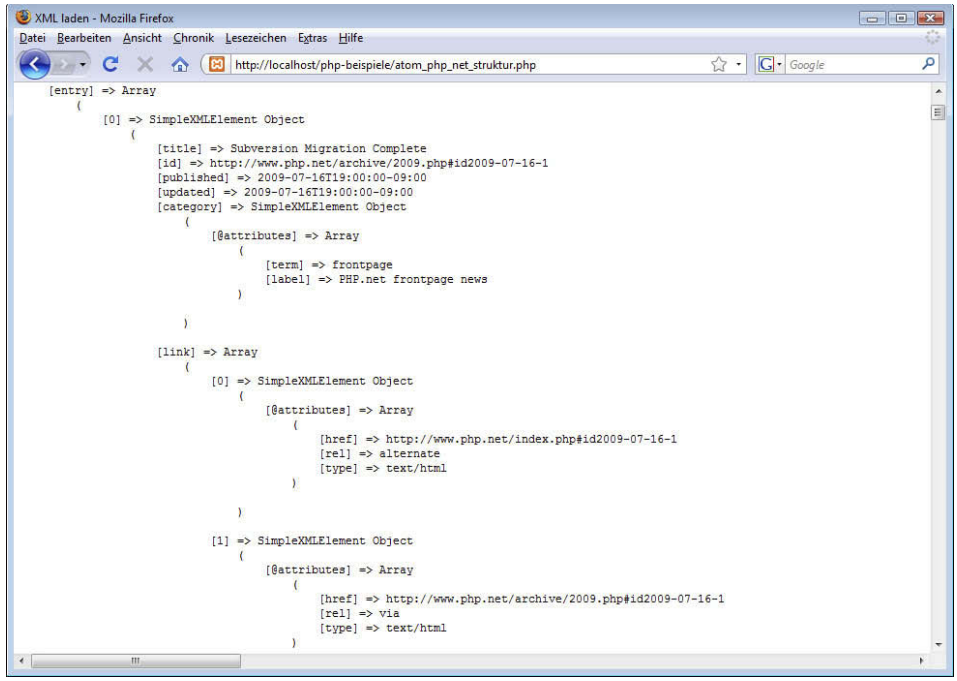


Abbildung 12.12: Ausschnitt aus der Ausgabe von atom_php_net_struktur.php

Tipp

`(string)` sollten Sie ebenfalls verwenden, wenn Sie andere Stringfunktionen auf SimpleXML-Objekte anwenden möchten oder wenn Sie einen Stringvergleich durchführen.

Listing 12.15: Die Titel mit Links der PHP.net-News werden als Liste ausgegeben (atom_php_net.php).

```
$xml = simplexml_load_file("http://www.php.net/feed.atom");
echo "<ul>\n";
foreach($xml->entry as $artikel) {
  $titel = htmlspecialchars(utf8_decode((string)$artikel->title));
  $link = htmlspecialchars(utf8_decode((string)$artikel->link["href"]));
  echo "<li><a href=\"$link\">$titel</a></li>\n";
}
echo "</ul>\n";
```

12.5 Phar-Archive

Phar-Archive ermöglichen es, eine gesamte PHP-Anwendung in einer Datei zusammenzufassen, um die Verteilung und Installation zu vereinfachen. Die Möglichkeit, Phar-Archive auszulesen, ist seit PHP 5.3 integriert.

12.5.1 Phar-Archive erstellen

Standardmäßig ist die *Erstellung* von Phar-Archiven allerdings aus Sicherheitsgründen deaktiviert. Das ist auch richtig so: Zumindest auf Produktivsystemen müssen Sie nur Phar-Archive lesen, sie aber nicht schreiben. Sehen wir uns trotzdem einmal an, wie Sie ein Phar-Archiv erstellen können.

Zuerst einmal müssen Sie hierfür in *php.ini* `phar.readonly` auf `Off` stellen, die *php.ini*-Datei speichern und den Webserver neu starten.

Nehmen wir an, Sie haben ein Unterverzeichnis im aktuellen Ordner mit Namen *pharvorlage*, das eine Datei mit dem in Listing 12.16 gezeigten Inhalt beinhaltet:

Listing 12.16: Im Beispiel ist es nur eine Datei, die im Ordner gespeichert ist, bei einer echten Anwendung wären es mehrere (index.php).

```
<?php
echo "hallo von phar";
?>
```

Dann genügt zum Erstellen des Phar-Archivs der in Listing 12.17 gezeigte Code:

Listing 12.17: Das Phar-Archiv wird erstellt (phar_erstellen.php).

```
$phar = new Phar("phar_beispiel.phar");
$phar->buildFromDirectory("pharvorlage");
```

Zuerst wird mit `new Phar()` ein neues Objekt der Phar-Klasse erstellt und dabei der Name der Datei übergeben, unter der das Phar-Archiv gespeichert werden soll. Der Methode `buildFormDirectory()` übergeben Sie den Namen des Verzeichnisses, in dem sich die Dateien befinden, die im Archiv aufgenommen werden sollen.

Wenn Sie danach in Ihr aktuelles Verzeichnis schauen, finden Sie eine neue Datei mit dem Namen *phar_beispiel.phar* vor und wissen, dass das Archiv angelegt wurde.

12.5.2 Phar-Archive benutzen

Das erstellte Phar-Archiv können Sie wie jede andere Bibliothek auch benutzen. Wenn Sie es beispielsweise per `include` einbinden, wird die *index.php*-Datei des Archivs eingebunden:

Listing 12.18: Das Phar-Archiv kann per include eingebunden werden (phar_nutzen.php).

```
include "phar_beispiel.phar";
```

Mehr Optionen stehen zur Verfügung, wenn Sie mit der `Phar`-Klasse arbeiten. In der Dokumentation der `Phar`-Klasse, sehen Sie, dass diese die Klasse `DirectoryIterator` erweitert.

```
Phar extends DirectoryIterator
```

`DirectoryIterator` liefert Ihnen nützliche Informationen über Dateien.

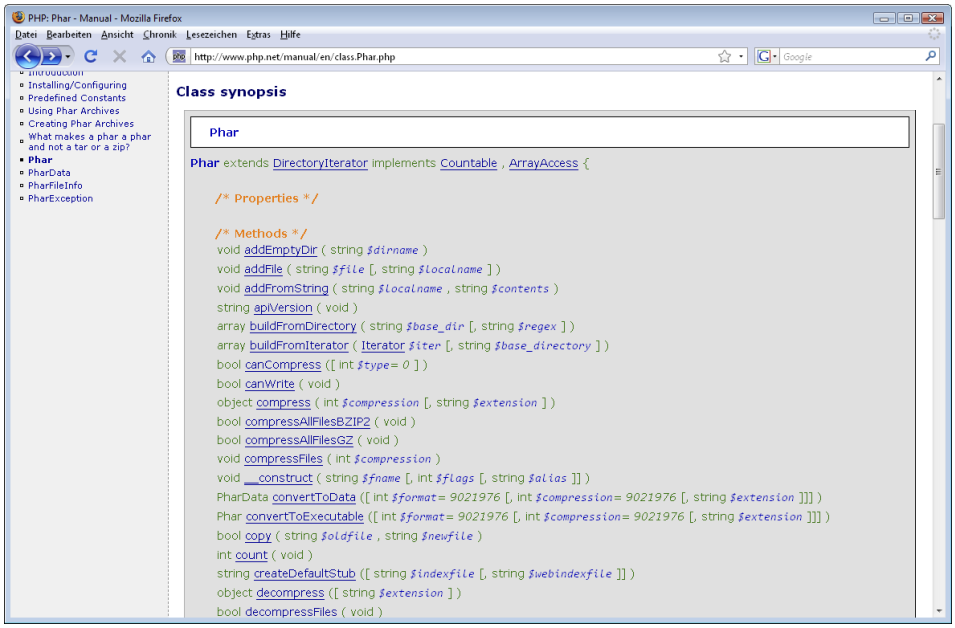


Abbildung 12.13: Einen Überblick über den Aufbau der Klasse Phar inklusive aller Methoden liefert das PHP-Manual unter http://www.php.net/manual/en/class.Phar.php.

Hierzu ein Beispiel.

Listing 12.19: Informationen über die Dateien des Phar-Archivs auslesen (phar_nutzen_2.php)

```
$phar = new Phar("phar_beispiel.phar", 0)
foreach ($phar as $datei) {
  echo $datei->getFilename() . "<br/>";
  echo htmlspecialchars(file_get_contents($datei->getPathName()));
}
```

Zuerst wird ein neues Objekt der Phar-Klasse erstellt. Danach wird dieses in einer Schleife durchlaufen, da es ja im Normalfall mehrere Dateien enthält. Der Dateiname wird über `getFilename()` ausgelesen, und schließlich werden die Inhalte ausgegeben.

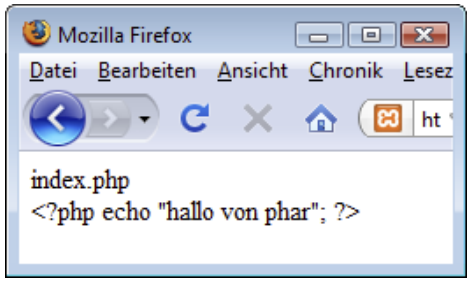


Abbildung 12.14: Ausgabe von Listing 12.19

Bisher haben Sie gelesen, wie PHP mit Datenbanken und Dateien umgehen kann. Im nächsten Kapitel wird's bunter: Da geht's um Bilder.

13 Mit Grafiken arbeiten

Bisher haben wir im Wesentlichen immer Texte ausgeben lassen. Mit PHP können Sie aber auch ganz wunderbar Grafiken bearbeiten: Sie erfahren in diesem Kapitel, wie die Grafikbearbeitung prinzipiell funktioniert und sehen an einem Beispiel, wie Sie Thumbnails erstellen und dynamische Diagramme generieren können.

13.1 Bildbearbeitung mit PHP – Grundlegendes

Es gibt verschiedene Erweiterungen für die Arbeit mit Grafiken. *Exif* ist für die Arbeit mit den EXIF-Metainformationen von Grafiken gedacht, *ImageMagick* und *GD* sind für die Bearbeitung von Grafiken vorgesehen. Hier soll die Bildbearbeitung und -erzeugung mit der Standarderweiterung *GD* gezeigt werden.

Tipp

Zuerst einmal sollten Sie überprüfen, ob GD installiert ist. Dies verrät wieder ein Blick in die Ausgabe von `phpinfo()`. Suchen Sie hier nach *gd*. Wie Sie die GD-Erweiterung installieren, finden Sie unter *http://www.php.net/manual/de/image.installation.php*.

13.1.1 Einfache Bilder erstellen

Zur Erstellung von Bildern brauchen Sie eine PHP-Datei, die nur PHP-Code enthält – also kein (X)HTML-Grundgerüst: Sie dient ja allein der Erzeugung der Grafik, im Beispiel einer JPEG-Grafik.

Das Skript ist kurz (Listing 13.1).

Listing 13.1: Ein JPEG-Bild wird erstellt (bild_erstellen.php).

```
<?php
header ("Content-type: image/jpg");
$bild = imagecreatetruecolor(100, 50);
imagejpeg($bild);
imagedestroy($bild);
?>
```

Zuerst wird der Header mit dem richtigen MIME-Typ definiert.

Achtung

Vor der Zeile mit dem Header darf nichts ausgegeben werden. Sonst erhalten Sie die Fehlermeldung »Header already sent ...« (siehe hierzu auch Kapitel 8).

Das Bild selbst erstellen Sie mit der Funktion `imagecreatetruecolor()`. Sie erwartet als Parameter die Breite und Höhe des Bilds in Pixeln. `imagecreatetruecolor()` gibt ein Handle auf das Bild zurück, über das Sie das Bild dann im Weiteren ansprechen können. Schließlich dient die Funktion `imagejpeg()` zur Erstellung der JPEG-Grafik und `imagedestroy()` gibt den Speicher wieder frei. Beide erwarten den Handle des Bilds als Parameter.

Jetzt benötigen wir noch ein weiteres Skript, in dem die Grafik angezeigt wird – ein ganz normales XHTML-Dokument, ganz ohne PHP-Code. In diesem wird die Grafik über das `img`-Element eingebunden. Als Pfad wird bei `src` der *Name der PHP-Datei* angegeben.

Listing 13.2: Das per PHP erstellte Bild wird in einer XHTML-Datei eingebunden (bild_ausgeben.html).

```
<img src="bild_erstellen.php" alt="" />
```

Zugegebenermaßen ist das Bild selbst noch nicht sehr ausgefeilt: Es ist einfach ein schwarzes Rechteck in der angegebenen Größe.

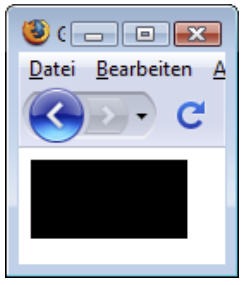


Abbildung 13.1: Das per PHP erstellte »Bild«

Im bisherigen Beispiel wird das Bild über das PHP-Skript ausgegeben und um es anzuzeigen, muss das PHP-Skript eingebunden werden. Sie können das Bild aber

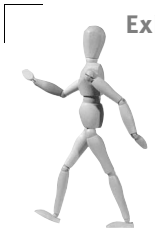
auch direkt als JPEG abspeichern. Hierfür geben Sie als zweiten Parameter von `imagejpeg()` den Bildnamen an, unter dem es gespeichert werden soll (Listing 13.3).

Listing 13.3: Jetzt wird das Bild abgespeichert (bild_erstellen2.php).

```
<?php
$bild = imagecreatetruecolor(300, 400);
imagejpeg($bild, "bild.jpg");
imagedestroy($bild);
?>
```

Das werden wir brauchen, wenn wir in Abschnitt 13.2 Thumbnails von Bildern aus einem Ordner erstellen lassen.

Wenn Sie dieses Skript aufrufen, erzeugt es im Browser keine sichtbare Ausgabe, aber danach sollten Sie in Ihrem Ordner das Bild *bild.jpg* vorfinden.



Exkurs: Grafikformate

In JPEG-Bildern können Sie über 16 Mio. Farben darstellen lassen. JPEG eignet sich damit besonders gut für Fotos. Alternativen dazu sind GIF mit nur 256 Farben für flächige Grafiken gedacht. PNG bietet eine verlustfreie Komprimierung und unterstützt genauso wie JPEG True Color, d. h. über 16 Mio. Farben, zumindest in der Version PNG 24. Daneben gibt es auch PNG 8, das nur 8-Bit-Grafiken mit 256 verschiedenen Farben erlaubt. Zum Erstellen von GIF oder PNG gibt es parallel zu `imagejpeg()` die Funktionen `imagegif()` und `imagepng()`.

Mit verschiedenen Funktionen können Sie auf Ihrem Bild Formen zeichnen. Dafür müssen Sie erst einmal Farben zuweisen. Hierzu dient die Funktion `imagecolorallocate()`, der Sie als ersten Parameter den Bild-Handle übergeben und dahinter die gewünschte RGB-Farbe. Sie können die Werte für Rot, Grün und Blau dezimal oder auch hexadezimal angeben.

```
$weiss = imagecolorallocate($bild, 255, 255, 255);
```

Die Farbe können Sie zum Beispiel bei der Funktion `imagefilledrectangle()` einsetzen, mit der sich ein gefülltes Rechteck zeichnen lässt:

```
imagefilledrectangle($bild, 10, 10, 100, 50, $weiss);
```

Diese Funktion erwartet als ersten Parameter den Bild-Handle, dann X- und Y-Koordinate der oberen linken Ecke des Bilds, und die X- und Y-Koordinate der unteren rechten Ecke sowie schließlich die Farbe. Parallel zu `imagefilledrectangle()` gibt es `imagerectangle()`, das ein Rechteck ohne Füllung nur mit Konturen erstellt.

Auch Text kann ausgegeben werden. Hierfür dient die Funktion `imagestring()`.

```
imagestring($bild, 5, 30, 15, $text, $weiss);
```

Sie erwartet als ersten Parameter wieder den Bild-Handle, der zweite Parameter benennt die Nummer des Fonts. Dies kann eine Zahl von 1 bis 5 sein, dann wird einer der integrierten Fonts genommen. Die beiden nächsten Parameter benennen die X- und die Y-Koordinate der linken oberen Ecke. Folgen noch der Text, der ausgegeben werden soll, und die Farbe, in der der Text geschrieben werden soll.

Im folgenden Listing wird ein Rechteck gezeichnet und darauf ein Text gesetzt.

Listing 13.4: Text in Grafik (bild_mit_text.php)

```
01 <?php
02 header ("Content-type: image/png");
03 $bild = imagecreatetruecolor(200, 50);
04 $weiss = imagecolorallocate($bild, 255, 255, 255);
05 $rot = imagecolorallocate($bild, 255, 0, 0);
06 $text = "Grafik mit Text";
07 imagefilledrectangle($bild, 10, 10, 190, 40, $rot);
08 imagestring($bild, 5, 30, 15, $text, $weiss);
09 imagepng($bild);
10 imagedestroy($bild);
11 ?>
```

Sie sehen, dieses Mal wird ein PNG-Bild erstellt und der entsprechende Header gesendet (Zeile 2). Ein Bild mit 200 Pixeln Breite und 50 Pixeln Höhe wird erzeugt (Zeile 3) und zwei Farben werden definiert (Zeile 4 und 5). `imagefilledrectangle()` in Zeile 7 zeichnet das Rechteck, `imagestring()` ergänzt den Text (Zeile 8). Schließlich wird das PNG-Bild ausgegeben und der Bild-Handle wieder freigegeben.

Abbildung 13.2: Text auf der Grafik

Hinweis

Sie können neben Rechtecken auch andere Formen zeichnen: `imageellipse()` zeichnet eine Ellipse, `imagearc()` ein Bogenstück, `imagepolygon()` ein Polygon. Zu allen dreien gibt es die ausgefüllten Varianten: `imagefilledellipse()`, `imagefilledarc()` oder `imagefilledpolygon()`. Diese und weitere Funktionen finden Sie im Manual unter *http://www.php.net/manual/de/book.image.php*.

13.2 Vorschaubilder per PHP erzeugen

Wenn Sie viele Bilder auf einer Webseite präsentieren möchten, bietet es sich an, zuerst einmal zur Verringerung der Ladezeit kleine Vorschaubilder anzuzeigen. Interessiert sich jemand für das Bild, kann er sich das größere Original ansehen. Um Vorschaubilder automatisch zu erstellen, muss man per PHP bereits vorhandene Bilder bearbeiten, was wir uns jetzt ansehen.

Abbildung 13.3 zeigt, wie es aussehen soll: Von allen Bildern aus einem vorgegebenen Ordner werden automatisch Vorschaubilder erzeugt und angezeigt.

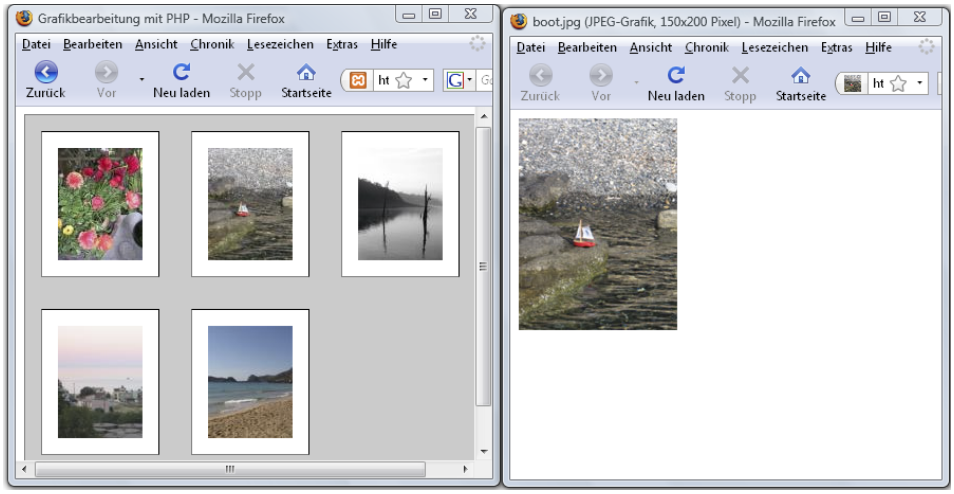


Abbildung 13.3: Links: Kleine Vorschaubilder. Wenn man auf eines klickt, wird die größere Version angezeigt (rechts).

Im Beispiel gibt es zwei Dateien:

- *vorschaubilder.php* ist für die Erstellung der Vorschaubilder verantwortlich.
- *vorschaubilder_ausgeben.php* sorgt für die Ausgabe der Vorschaubilder.

Außerdem gibt es zwei Ordner, die beide vorhanden sein müssen.

- *bilder* enthält die Bilder, von denen Vorschaubilder erzeugt werden sollen.
- *vorschaubilder* soll später die kleinen Bilder aufnehmen, ist zu Beginn jedoch noch leer.

Erst einmal zum Skript *vorschaubilder.php*. Dieses liest alle Bilder aus dem Unterordner *bilder* aus, erstellt die Thumbnails und speichert sie im Unterordner *vorschaubilder*.

Am Anfang steht das Auslesen aller JPEG-Bilder aus dem Ordner *bilder*:

```
01 $bv = "bilder";
02 $vb = "vorschaubilder";
03 $verzeichnis = opendir($bv);
04 $bilder = array();
05 while($datei = readdir($verzeichnis)) {
06   if (preg_match("/\.jpe?g$/", $datei)) {
07     $bilder[] = $datei;
08   }
09 }
10 closedir($verzeichnis);
```

Der Ordner wird über `opendir()` in Zeile 3 geöffnet. Der Inhalt wird mit `readdir()` in einer `while`-Schleife ausgelesen (Zeile 5). Ist keine Datei mehr vorhanden, gibt `readdir()` `false` zurück und damit wird die Schleife beendet.

Ob es sich beim jeweiligen Dateinamen um ein JPEG-Bild handelt, prüft der reguläre Ausdruck in Zeile 6:

```
"/\.jpe?g$/"
```

Er bedeutet, dass zuerst ein Punkt erforderlich ist (`\.`), dem `jpeg` oder `jpg` folgt (`jpe?g`), die sich am Ende befinden müssen – der Ausdruck ist über das `$`-Zeichen am Ende verankert.

Wenn der Dateiname auf das Muster passt, wird er als Arrayelement dem Array `$bilder` hinzugefügt (Zeile 7). Danach wird das Verzeichnis über `closedir()` wieder geschlossen (Zeile 10).

Jetzt geht es an die Erstellung der Thumbnails. Dabei werden vier neue Funktionen eingesetzt: `imagecreatefromjpeg()` dient zum Öffnen eines bereits vorhandenen JPEG-Bilds, über `imagesx()` und `imagesy()` lesen Sie die Breite und die Höhe eines Bilds aus. Die Größe eines Bilds ändern Sie über `imagecopyresampled()`.

Listing 13.5: Die Vorschaubilder erzeugen (vorschaubilder.php)

```
11 foreach ($bilder as $bild) {
12   $b = imagecreatefromjpeg("$bv/$bild");
13   $originalbreite = imagesx($b);
14   $originalhoehe = imagesy($b);
15   $neuebreite = 80;
16   $neuehoehe = floor($originalhoehe * ($neuebreite / $originalbreite));
17   $neuesbild = imagecreatetruecolor($neuebreite, $neuehoehe);
18   imagecopyresampled($neuesbild, $b, 0, 0, 0, 0, $neuebreite, $neuehoehe,
       $originalbreite, $originalhoehe);
19   echo "Thumbnail erzeugt für $bild<br />";
20   imagejpeg($neuesbild, "$vb/$bild");
21   imagedestroy($neuesbild);
22 }
```

Über eine `foreach`-Schleife (Zeile 11) werden alle Elemente des `$bilder`-Arrays durchlaufen. In Zeile 12 wird über `imagecreatefromjpeg()` das Bild eingelesen und in Zeile 13 und 14 werden seine Ausmaße ermittelt. Als neue Breite des Thumbnails werden in Zeile 15 80 Pixel festgelegt – das können Sie natürlich an Ihren Bedarf anpassen. Da die Bilder proportional skaliert werden können, wird die neue Höhe in Zeile 16 proportional basierend zur ursprünglichen Höhe berechnet und der Wert mit `floor()` gerundet.

Zeile 17 erstellt ein neues Bild-Handle mit den neu berechneten Werten. Das Bild wird dann über `imagecopyresampled()` erzeugt. Sehen wir uns diese Zeile einmal genau an:

```
18   imagecopyresampled($neuesbild, $b, 0, 0, 0, 0, $neuebreite, $neuehoehe,
       $originalbreite, $originalhoehe);
```

`imagecopyresampled()` erwartet als ersten Parameter den neuen Bild-Handle, als zweiten den Bild-Handle des ursprünglichen Bilds. Die nächsten vier Parameter bestimmen die Koordinaten der oberen linken Ecke des neuen und des ursprünglichen Bilds: Sie könnten über `imagecopyresampled()` nämlich auch einen Teil aus einem Bild ausschneiden. Über `0, 0, 0, 0` wird dafür gesorgt, dass das gesamte Bild genommen wird. Schließlich erwartet die Funktion dann noch die Angabe von Breite und Höhe des neuen sowie des ursprünglichen Bilds.

Zeile 19 gibt eine Meldung aus, welches Thumbnail erzeugt wurde. In Zeile 20 wird das neue Bild mit dem ursprünglichen Namen im Ordner für die Vorschaubilder abgespeichert.

Achtung

Beachten Sie, dass, um das Beispiel kompakt zu halten, beim Beispiel keine Überprüfungen auf mögliche Fehler durchgeführt werden.

Es fehlt noch die Ausgabe der Vorschaubilder, die ebenfalls automatisch erfolgen soll: In der Datei *vorschaubilder_ausgeben.php* werden zuerst wieder alle JPEG-Bilder ausgelesen:

```
01 $bv = "bilder";
02 $verzeichnis = opendir($bv);
03 $bilder = array();
04 while($datei = readdir($verzeichnis)) {
05   if (preg_match("/\.jpe?g$/", $datei)) {
06     $bilder[] = $datei;
07   }
08 }
09 closedir($verzeichnis);
```

Für jedes dort gefundene Bild wird in einer `foreach`-Schleife das Bild angezeigt und mit einem Link auf das große Bild ausgegeben:

Listing 13.6: Vorschaubilder anzeigen (vorschaubilder_ausgeben.php)

```
10 foreach($bilder as $bild) {
11   echo "<a href='bilder/$bild'><img src='vorschaubilder/$bild' alt='' /></a>\n";
12 }
```

Der dadurch erzeugte XHTML-Code sieht beispielsweise so aus:

```
<a href='bilder/blumen.jpg'><img src='vorschaubilder/blumen.jpg' alt='' /></a>
```

Die Anordnung der Bilder könnte über eine Tabelle geschehen – eleganter ist es, dieses wie im Beispiel über CSS zu realisieren.

Tipp

Wenn Sie das Beispiel ausprobieren und auf eines der Vorschaubilder klicken, so wird die größere Version direkt im selben Browserfenster angezeigt. Soll dieses hingegen in einem kleineren Browserfenster in festgelegter Größe angezeigt werden, so lässt sich das über JavaScript realisieren (siehe hierzu *http://de.selfhtml.org/javascript/objekte/window.htm#open*).

13.3 Diagramme erstellen

Die Grafikbearbeitungsfunktionen lassen sich auch zur Erstellung von dynamischen Diagrammen verwenden, beispielsweise um den aktuellen Stand einer Umfrage zu visualisieren. Solche Diagramme können Sie im Prinzip selbst erstellen: mithilfe der vorgestellten Funktionen wie `imagefilledrectangle()` für ein Balkendiagramm oder `imagefilledarc()` für ein Tortendiagramm. Bei einem einfachen Diagramm ist das noch einfach zu machen, für komplexere Diagramme inklusive Beschriftung und Achsen wird's dann schon aufwändiger. Da hilft die Bibliothek JpGraph (*http://www.aditus.nu/jpgraph/*) weiter.

Hinweis

JpGraph ist für den nichtkommerziellen Einsatz kostenlos, für die kommerzielle Nutzung müssen Sie eine Lizenz erwerben (*http://www.aditus.nu/jpgraph/proversion.php*).

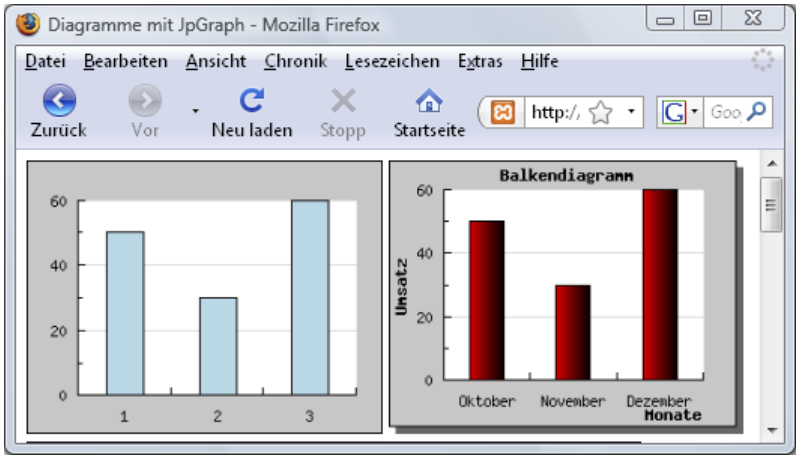


Abbildung 13.4: Balkendiagramme über JpGraph

Um JpGraph zu benutzen, müssen Sie es erst unter *http://www.aditus.nu/jpgraph/jpdownload.php* herunterladen und entpacken. Dieses Paket beinhaltet zwei Ordner: *docs* für die Dokumentation und *src* für die Dateien, die Sie dann einbinden müssen.

13.3.1 Balkendiagramme

Erstellen wir erst einmal das in Abbildung 13.4 links dargestellte Balkendiagramm.

Listing 13.7: Ein einfaches Balkendiagramm (balkendiagramm_1.php)

```
01 <?php
02 header ("Content-type: image/png");
03 require "jpgraph/src/jpgraph.php";
04 require "jpgraph/src/jpgraph_bar.php";
05
06 $daten = array(50, 30, 60);
07 $graph = new Graph(260, 200, "PNG");
08 $graph->SetScale("textlin");
09 $bplot = new BarPlot($daten);
10 $graph->Add($bplot);
11 $graph->Stroke();
12 ?>
```

Der Code zur Erstellung des Diagramms steht in einer puren PHP-Datei ohne (X)HTML-Grundgerüst. Zuerst (Zeile 2) wird der erforderliche Header erstellt. Dann werden in Zeile 3 und 4 die benötigten Dateien eingebunden – hier müssen Sie bei Bedarf den Pfad noch anpassen.

Tipp

Die Datei *jpgraph.php* wird bei allen Diagrammtypen benötigt, *jpgraph_bar.php* hingegen nur bei Balkendiagrammen.

In Zeile 6 stehen die Daten, die dargestellt werden sollen, in Form eines Arrays.

Tipp

Diese Daten könnten aber selbstverständlich auch aus einer Datenbank stammen.

Zeile 7 erstellt eine neue Grafik, in Klammern werden die Breite, die Höhe und das Format übergeben. Mit der Methode `SetScale()` wird die Skala für X- und Y-Achse bestimmt: `textlin` bedeutet, dass die X-Achse Text enthält, die Y-Achse hingegen linear ist. Mit `new BarPlot()` wird ein neues Balkendiagramm erstellt und ihm gleichzeitig die Daten übergeben. Jetzt muss mit der Methode `Add()` noch das Diagramm dem Bild hinzugefügt werden (Zeile 10) und über die Methode `Stroke()` wird das Diagramm gezeichnet.

Das Diagramm lässt sich jetzt noch verfeinern (Abbildung 13.4, rechts).

Listing 13.8: Erweitertes Balkendiagramm (balkendiagramm_2.php)

```
01 <?php
02 header ("Content-type: image/png");
03 include "jpgraph/src/jpgraph.php";
04 include "jpgraph/src/jpgraph_bar.php";
05
06 $daten = array(50, 30, 60);
07 $xachse = array("Oktober", "November", "Dezember");
08 $graph = new Graph(260, 200, "auto");
09 $graph->SetScale("textlin");
10 $graph->SetShadow();
11 $graph->img->SetMargin(40, 30, 20, 40);
12 $bplot = new BarPlot($daten);
13 $bplot->SetFillgradient("red", "black", GRAD_VER);
14 $graph->Add($bplot);
15 $graph->xaxis->SetTickLabels($xachse);
```

```
16 $graph->title->Set("Balkendiagramm");
17 $graph->xaxis->title->Set("Monate");
18 $graph->yaxis->title->Set("Umsatz");
19 $graph->title->SetFont(FF_FONT1, FS_BOLD);
20 $graph->yaxis->title->SetFont(FF_FONT1, FS_BOLD);
21 $graph->xaxis->title->SetFont(FF_FONT1, FS_BOLD);
22 $graph->Stroke();
23 ?>
```

Die ersten Zeilen sind wie gehabt. In Zeile 7 wird ein zusätzliches Array mit den Beschriftungen der X-Achse definiert. Die Methode `SetShadow()` ergänzt einen Schatten um das Bild. Mit `SetMargin()` werden die Ränder festgelegt (Zeile 11). Für die Balken wird ein Farbverlauf bestimmt (Zeile 13), der von Rot zu Schwarz geht.

Nun werden diverse Beschriftungen gesetzt: In Zeile 15 erhalten die einzelnen Balken auf der X-Achse die Monatsnamen. Zeile 16 vergibt einen Titel für das gesamte Diagramm und Zeile 17 und 18 geben der X- und Y-Achse eine Beschriftung. Für die Beschriftungen wird über `SetFont()` eine von mehreren mitgelieferten Standardschriften gesetzt und als Schriftschnitt fett bestimmt.

13.3.2 Tortendiagramm

Die Erstellung des einfachen Tortendiagramms ist ganz ähnlich wie die des einfachen Balkendiagramms:

Listing 13.9: Ein einfaches Tortendiagramm (tortendiagramm_1.php)

```
01 <?php
02
03 header ("Content-type: image/png");
04 require "jpgraph/src/jpgraph.php";
05 require "jpgraph/src/jpgraph_pie.php";
06 require "jpgraph/src/jpgraph_pie3d.php";
07
08 $daten = array(50, 30, 60);
09 $graph = new PieGraph(450, 300, "PNG");
10 $p1 = new PiePlot3D($daten);
11 $graph->Add($p1);
12 $graph->Stroke();
13 ?>
```

Sie sehen, es werden für die 3D-Tortendiagramme neben der immer benötigten Datei *jpgraph.php* zwei weitere eingebunden (Zeile 5 und 6). Zur Erstellung des 3D-Tortendiagramms wird eine neue Instanz von `PieGraph` erstellt und dann eine neue Instanz von `PiePlot3D`. Wieder wird über `Add()` das Diagramm hinzugefügt und über `Stroke()` gezeichnet.

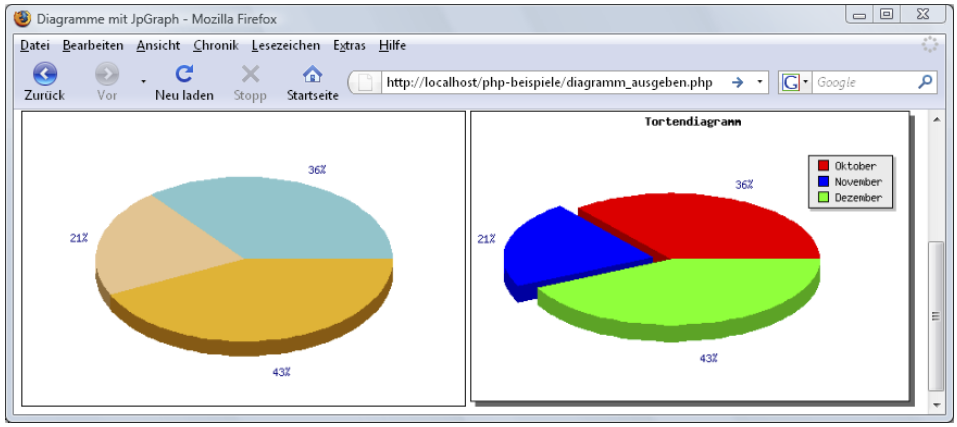


Abbildung 13.5: Auch 3D-Tortendiagramme lassen sich erstellen.

Das derzeitige Diagramm ist noch sehr schlicht und kann natürlich erweitert werden. Es soll aussehen wie in Abbildung 13.5, rechts. Die im Vergleich zur Basisversion geänderten Stellen sind hervorgehoben.

Listing 13.10: Erweitertes 3D-Tortendiagramm (tortendiagramm_2.php)

```
01 <?php
02
03 header ("Content-type: image/png");
04 require "jpgraph/src/jpgraph.php";
05 require "jpgraph/src/jpgraph_pie.php";
06 require "jpgraph/src/jpgraph_pie3d.php";
07
08 $daten = array(50, 30, 60);
09 $legende = array("Oktober", "November", "Dezember");
10 $graph = new PieGraph(450, 300, "PNG");
11 $graph->SetShadow();
12 $p1 = new PiePlot3D($daten);
13 $p1->ExplodeSlice(1);
14 $p1->SetCenter(0.45);
15 $p1->SetLegends($legende);
16 $p1->SetAngle(40);
17 $p1->SetSliceColors(array("red", "blue", "green"));
18 $graph->title->Set("Tortendiagramm");
19 $graph->title->SetFont(FF_FONT1,FS_BOLD);
20 $graph->Add($p1);
21 $graph->Stroke();
```

`ExplodeSlice()` (Zeile 13) bewirkt, dass ein Tortenstück etwas herausbewegt wird: In Klammern geben Sie an, welches, wobei die Zählung bei 0 beginnt. Über `SetCenter()` in Zeile 14 versetzen Sie das Zentrum des Diagramms, damit die Legende, die in Zeile

15 gesetzt wird, Platz findet. `SetAngle()` bestimmt, wie schräg das 3D-Diagramm angezeigt werden soll, die möglichen Werte liegen zwischen 10 und 80. Über `SetSliceColor()` legen Sie die Farben der Tortenabschnitte fest. Außerdem wird wieder ein Titel für das Diagramm und die Schrift bestimmt.

Dies ist natürlich nur ein kleiner Ausschnitt aus den Funktionen, die JpGraph bietet – weitere Informationen hierzu finden Sie in der bei JpGraph mitgelieferten Dokumentation. In dieser sind auch sehr viele Diagramme mit Beispielcode aufgeführt und außerdem gibt es eine Auflistung der einzelnen Klassen mit ihren Methoden und Eigenschaften.

14 Template-Engines am Beispiel von Smarty

Eine große Stärke von PHP ist, dass es sich so wunderbar in (X)HTML-Dateien integrieren und mit dem (X)HTML-Code kombinieren lässt – eine Eigenschaft, die gerade den Einstieg in PHP so einfach macht. Bei umfangreichen Projekten wird aber genau diese Mischung von (X)HTML und PHP-Code unübersichtlich. Zum einen ist der Code, der so gespickt mit (X)HTML-Tags ist, schwer zu warten, und zum anderen sind bei großen Projekten üblicherweise verschiedene Leute für die PHP-Programmierung und die (X)HTML/CSS-Realisierung zuständig. Und dann ist es auch besser, wenn diese an verschiedenen Dateien arbeiten.

Abhilfe schaffen hier Template-Systeme. Sie erlauben die Trennung von Programmierlogik und Ausgabe. Sie haben folgendes Prinzip: In der (X)HTML-Datei steht erst einmal nur (X)HTML mit einzelnen Template-Variablen. Diese Variablen werden dann dynamisch durch PHP ersetzt. Die Logik dazu wird jedoch in einer anderen Datei untergebracht. Das erleichtert ein verteiltes Arbeiten. Eine Person kümmert sich um den (X)HTML- (und CSS-)Teil, eine andere Person um die PHP-Programmierung. Wer das (X)HTML-Gerüst erstellt, muss nicht in die Tiefen von PHP eindringen.

Eines der bekanntesten Template-Systeme ist Smarty (*http://www.smarty.net/*) und soll hier vorgestellt werden. Ein Vorteil von Smarty ist, dass die Dateien nur einmal kompiliert werden müssen. Nur geänderte Templates werden erneut kompiliert. Außerdem können die Dateien bei Bedarf gecacht werden.

14.1 Erste Schritte mit Smarty

Bevor Sie erste Tests mit dem Template-System Smarty durchführen können, müssen Sie die Dateien von *http://www.smarty.net/download.php* herunterladen und extrahieren.

Jetzt benötigen Sie mehrere Ordner. Im Beispiel heißt der übergeordnete Ordner *smarty_beispiel*. In diesem befinden sich vier weitere Ordner: *cache*, *configs*, *templates*, *templates_c*.

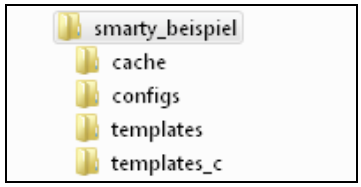


Abbildung 14.1: Ordner, die Sie für Smarty benötigen

- *cache* wird benötigt, falls Sie das Caching aktivieren.
- *configs* kann zusätzliche Konfigurationsdateien enthalten.
- *templates* ist der Ordner für die von Ihnen erstellten Template-Dateien.
- In *templates_c* speichert Smarty die kompilierten Templates.

Für die Ordner *cache* und *templates_c* benötigt Smarty Schreib- und Ausführrechte, zumindest 770 (zu den Rechten siehe auch Kapitel 12).

Aus den entpackten Smarty-Dateien benötigen Sie den Ordner *libs* samt Inhalt. Speichern Sie diesen auf derselben Ebene wie den Ordner *smarty_beispiel*.

Tipp

Im Beispiel werden alle Ordner innerhalb des Webroot-Ordners gespeichert. Wenn Sie bei Ihrem Provider auch Zugriff auf andere Verzeichnisse haben, können Sie diese auch außerhalb abspeichern.

Nun brauchen wir zuerst eine Datei, die definiert, wo sich die einzelnen Dateien befinden. Außerdem wird in dieser Datei eine neue Instanz der Smarty-Klasse erstellt.

Diese Datei wird später wo notwendig eingebunden. Das heißt: Nur diese Datei müssen Sie ändern, wenn Sie Ihre Dateien von Ihrem Testsystem auf das echte System hochspielen. Speichern Sie diese Datei im Ordner *smarty_beispiel*.

Listing 14.1: Pfade festlegen und neue Instanz der Smarty-Klasse erstellen (smarty_initialize.php)

```
01 <?php
02 date_default_timezone_set("Europe/Berlin");
03 define("SMARTY_DIR", "/xampp/htdocs/php-beispiele/kap_14/libs/");
04 require_once SMARTY_DIR."Smarty.class.php";
05 $smarty = new Smarty;
06 $smarty->template_dir = "/xampp/htdocs/php-
     beispiele/kap_14/smarty_beispiel/templates/";
```

```
07 $smarty->compile_dir = "/xampp/htdocs/php-
     beispiele/kap_14/smarty_beispiel/templates_c/";
08 $smarty->config_dir =  "/xampp/htdocs/php-
     beispiele/kap_14/smarty_beispiel/config/";
09 $smarty->cache_dir =     "/xampp/htdocs/php-
     beispiele/kap_14/smarty_beispiel/cache/";
10 //$smarty->caching = true;
11 ?>
```

Achtung

Bei diesem Beispiel wird davon ausgegangen, dass sich die Dateien im Verzeichnis */xampp/htdocs/php-beispiele/kap_14/* befinden. Das müssen Sie noch an Ihre Gegebenheiten anpassen.

Listing 14.1 zeigt das vollständige Beispiel – es wird kein (X)HTML-Code drumherum benötigt.

Da es sich um PHP-Code handelt, dürfen PHP-Start- und -End-Tag nicht fehlen. In Zeile 2 wird außerdem eine Default-Zeitzone gesetzt. Wichtig ist Zeile 3. Diese definiert die Konstante `SMARTY_DIR` und setzt sie auf den Pfad zum *libs*-Ordner der Smarty-Dateien. Zeile 4 bindet die Datei *Smarty.class.php* ein, die sich im *libs*-Ordner befindet. Die Pfade müssen natürlich hier stimmen.

Zeile 5 initialisiert ein neues Smarty-Objekt. Außerdem wird definiert, wo sich die einzelnen Ordner befinden. `template_dir` verweist auf das *templates*-Verzeichnis, `compile_dir` auf das *templates_c*-Verzeichnis. `config_dir` und `cache_dir` brauchen Sie nicht für die ersten Tests, können diese aber gleich schon einmal hier angeben.

Zeile 10 zeigt, wie man Caching aktivieren würde. Für die ersten Tests arbeiten Sie aber besser ohne Caching, sonst sehen Sie die Änderungen bei Tests nicht wie gewünscht.

Hinweis

Informationen zu den möglichen Einstellungen fürs Caching finden Sie unter *http://www.smarty.net/manual/de/caching.php*.

Achtung

Kontrollieren Sie noch einmal die Pfade – auch wenn das eigentlich banal ist, sind falsche Pfade die häufigsten Probleme am Anfang.

Speichern Sie diese Datei dann im Ordner *smarty_beispiele* ab.

Nun brauchen Sie *zwei* Dateien. Eine enthält die PHP-Anwendungslogik (*smarty_start.php*), die andere das Template – den (X)HTML-Code mit Platzhaltern (*index.tpl*).

Die Datei mit der PHP-Anwendungslogik kann sich irgendwo in Ihrem Webverzeichnis befinden. Diese Datei wird später aufgerufen.

Listing 14.2: Die Anwendungslogik (smarty_start.php)

```
<?php
require_once "smarty_beispiel/smarty_initialize.php";
$meintitel = "Ein erster Test mit Smarty";
$meinestadt = "München";
$smarty->assign("neuertitel", $meintitel);
$smarty->assign("stadt", $meinestadt);
$smarty->display("index.tpl");
?>
```

Zuerst wird die Initialisierungsdatei per `require_once` eingebunden, hier müssen Sie den Pfad entsprechend anpassen. Außerdem werden zwei Variablen `$meintitel` und `$meinestadt` erstellt und mit Textinhalt gefüllt. Im einfach gehaltenen ersten Beispiel ist es statischer Text – aber die Inhalte könnten natürlich genauso gut das Ergebnis einer komplexen Berechnung sein oder aus einer Datenbank stammen.

Mit der Methode `$smarty->assign()` werden die Variablen Platzhaltern zugeordnet. `neuertitel` und `stadt` sind zwei Platzhalter, die Ihnen gleich in der Template-Datei *index.tpl* noch begegnen werden. Schließlich wird über `$smarty->display()` die Template-Datei bestimmt, die ausgegeben werden soll. Im Beispiel ist es *index.tpl*.

Die Datei *index.tpl muss sich im templates-Ordner* befinden. Sie hat folgenden Inhalt:

Listing 14.3: Die Template-Datei mit den Platzhaltern (index.tpl)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
```

```
  <title>{$neuertitel}</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
Ich wohne in {$stadt}.
</body>
</html>
```

Die *index.tpl*-Datei enthält ganz normales XHTML. Zusätzlich werden jedoch zwei Platzhalter eingefügt: `{$neuertitel}` und `{$stadt}`. Der Inhalt dieser Platzhalter wurde durch `$smarty->asign()` in der *smarty_start.php*-Datei zugewiesen:

```
$smarty->assign("neuertitel", $meintitel);
$smarty->assign("stadt", $meinestadt);
```

Wenn Sie jetzt *smarty_start.php* aufrufen, erhalten Sie das in Abbildung 14.2 gezeigte Ergebnis. Der Seitentitel und die Angabe der Stadt sind eingefügt.

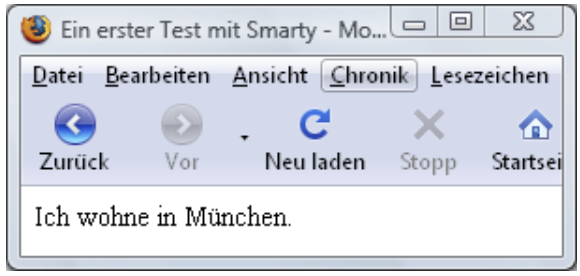


Abbildung 14.2: Das erste Smarty-Beispiel

Abbildung 14.3 zeigt noch einmal alle beteiligten Dateien: Oben links ist *smarty_start.php*, darunter *index.tpl*: *smarty_start.php* beinhaltet die ganze Anwendungslogik und sorgt dafür, dass *index.tpl* verarbeitet wird. *index.tpl* beinhaltet normales XHTML mit zwei Platzhaltern. Im Browser aufgerufen wird *smarty_start.php*. Das, was man dann im Browser sieht, ist normales XHTML, wo die Platzhalter aus *index.tpl* auf die in *smarty_start.php*-definierte Weise gefüllt sind. Den Quellcode des Ergebnisses sehen Sie rechts abgebildet.

14.2 Eine eigene Smarty-Klasse

Die ganze Konfiguration lässt sich elegant durch die Erstellung einer von Smarty abgeleitete Klasse definieren, die ebenfalls im Ordner *smarty_beispiel* abgespeichert wird.

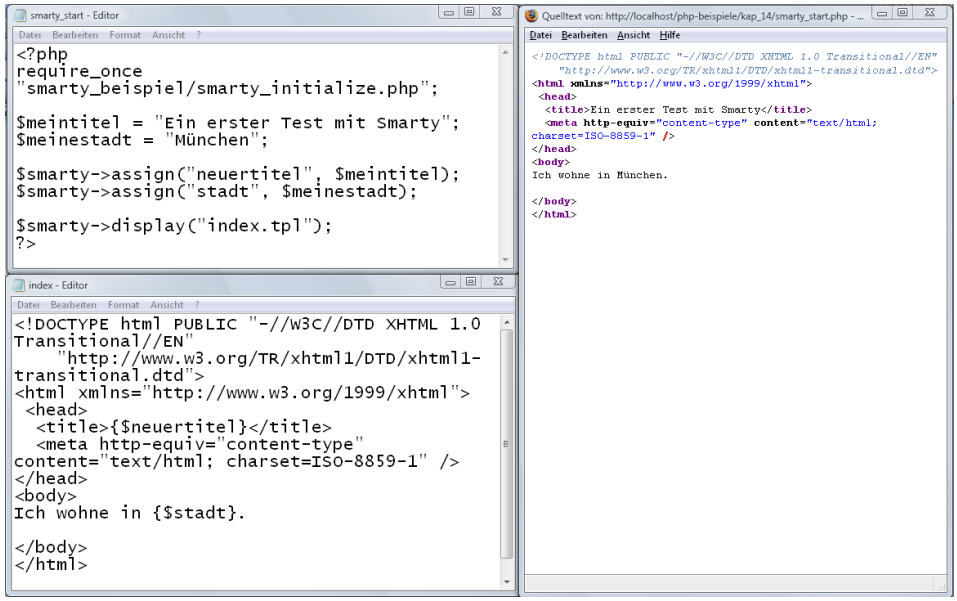


Abbildung 14.3: Die zwei Erstellungsdateien links und das Ergebnis rechts

Listing 14.4: Die von Smarty abgeleitete Klasse MeinSmarty (meinsmarty.php)

```
01 <?php
02 date_default_timezone_set("Europe/Berlin");
03 define("SMARTY_DIR", "/xampp/htdocs/php-beispiele/kap_14/libs/");
04 require_once SMARTY_DIR."Smarty.class.php";
05 class MeinSmarty extends Smarty
06 {
07   public function __construct()
08   {
09     parent::__construct();
10     $this->template_dir = "/xampp/htdocs/php-
         beispiele/kap_14/smarty_beispiel/templates/";
11     $this->compile_dir =   "/xampp/htdocs/php-
         beispiele/kap_14/smarty_beispiel/templates_c/";
12     $this->config_dir =     "/xampp/htdocs/php-
         beispiele/kap_14/smarty_beispiel/config/";
13     $this->cache_dir =     "/xampp/htdocs/php-
         beispiele/kap_14/smarty_beispiel/cache/";
14     //$this->caching = true;
15   }
16 }
17 ?>
```

Hier wird jetzt eine eigene Klasse `MeinSmarty` definiert, die von der Klasse `Smarty` abgeleitet ist. Damit die folgenden Konfigurationseinstellungen automatisch ausgeführt werden, werden sie in den Konstruktor verlagert. Die Konstruktormethode `__construct()` wird ja automatisch bei der Erstellung eines neuen Objekts aufgerufen. Zuerst wird in der Konstruktormethode jedoch der Konstruktor der Elternklasse über `parent::__construct()` aufgerufen. Dann wird über `$this->template_dir` das Verzeichnis für die Templates gesetzt und ebenfalls die anderen Verzeichnisse.

Erstellen wir eine neue Datei zum Aufruf mit Namen `smarty_oo.php`., die die neu definierte Klasse nutzt. Hier ändert sich nicht viel: Zuerst muss die Datei mit der Klasse eingebunden werden und dann eine neue Instanz von `MeinSmarty` erstellt werden. Wieder soll als Template die Datei *index.tpl* genutzt werden und auch die Ausgabe ist dieselbe wie in Abbildung 14.2 gezeigt.

Listing 14.5: Dieses Mal wird die abgeleitete Klasse MeinSmarty benutzt (smarty_oo.php).

```
<?php
require_once "smarty_beispiel/meinsmarty.php";
$smarty = new MeinSmarty;
$meintitel = "Ein erster Test mit Smarty";
$meinestadt = "München";
$smarty->assign("neuertitel", $meintitel);
$smarty->assign("stadt", $meinestadt);
$smarty->display("index.tpl");
?>
```

14.3 Weitere Möglichkeiten von Smarty

Bisher haben wir uns in den Beispielen auf das prinzipielle Zusammenspiel von Smarty-Datei und Template-Datei konzentriert. Jetzt zeigen ein paar weitere Beispiele, was mit der Template-Engine alles möglich ist.

Da immer mehrere Dateien mitspielen, sind im Folgenden die Beispiele jeweils in Form einer Tabelle gezeigt. Links sehen Sie den Code in der Anwendungslogik-Datei, in der Mitte die entsprechende Codezeile in der Template-Datei und rechts ist aufgeführt, welche Ausgabe das bewirkt. Damit das wie angegeben klappt, muss vor dem bei *smarty_oo_erweitert.php* angegebenen Code natürlich die Datei mit der Klassendefinition per `require_once` eingebunden sein und eine neue Instanz der Klasse `MeinSmarty` erstellt sein. In *beispiel_erweitert.tpl* befindet sich natürlich außer den Codebeispielen ein vollständiges (X)HTML-Grundgerüst.

Das erste Beispiel zeigt, wie man die zufällige Ausgabe eines Bilds mit Smarty realisiert – Sie kennen das Beispiel aus Kapitel 4.

Tipp

Am Ende des Abschnitts finden Sie das Beispiel im Gesamtzusammenhang.

In *smarty_oo_erweitert.php* wird ein Array mit Bilddateien erstellt, eines zufällig ausgewählt und dann der Platzhalter `bild` mit dem zufälligen Bild verknüpft. In *beispiel_erweitert.tpl* genügt es dann, diesen als Wert des `src`-Attributs eines `img`-Tags zu benutzen. (Im Beispiel wird der Ordnername `bilder/` vorher ergänzt, da sich die Bilder in einem eigenen Ordner befinden.)

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe

`$bilder = array("blumen.jpg", "boot.jpg", "landschaft.jpg", "stadt_am_meer.jpg", "strand.jpg");`
`$max = count($bilder)-1;`
`$zufallszahl = rand(0, $max);`
`$meinbild = $bilder[$zufallszahl];`
`$smarty->assign("bild", $meinbild);`
	`<img src="bilder/{$bild}" />`	Ein zufälliges Bild wird angezeigt.

Tabelle 14.1: Ein zufällig gewähltes Bild anzeigen lassen

Tipp

Smarty kann Ihnen auch das `img`-Element samt korrekter `width`- und `height`-Attribute erzeugen. Schreiben Sie hierfür in die Template-Datei:

```
{html_image file="bilder/$bild"}
```

Sie können in der Template-Datei auch auf Arrays zugreifen. Bei indizierten Arrays schreiben Sie den Index in eckigen Klammern.

smarty_oo_erweitert.php definiert ein Beispielarray mit drei Elementen.

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe
`$meineliste = array("Sonne", "Mond", "Sterne");` `$smarty->assign("liste", $meineliste);`	`{$liste[0]}`	Sonne

Tabelle 14.2: Ein Element eines indizierten Arrays ausgeben lassen

Selbstverständlich können Sie in der Template-Datei auch auf assoziative Arrays zugreifen. Dabei schreiben Sie den Schlüssel hinter den Variablennamen durch einen Punkt getrennt.

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe
`$meinefarben = array("red"=>"rot", "green"=>"grün");` `$smarty->assign("farbe", $meinefarben);`	`{$farbe.red}`	rot

Tabelle 14.3: Ein Element aus einem assoziativen Array ausgeben lassen

Der Inhalt von Variablen lässt sich über so genannte Variablenmodifikatoren ändern. Mit `truncate` schneiden Sie Textinhalt nach einer bestimmten Anzahl von Zeichen ab. Dies ist beispielsweise praktisch, wenn Sie auf der Startseite einen Teaser eines längeren Textes mit einer vordefinierten Länge erzeugen möchten.

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe
`$meinemeldung = "Aufgrund des schönen Wetters ändern wir unsere Pläne. Wir gehen ins Schwimmbad statt ins Kino";` `$smarty->assign("meldung", $meinemeldung);`	`{$meldung|truncate:40:" ..."}`	Aufgrund des schönen Wetters ändern ...
	`{$meldung|truncate:40:"...":true}`	Aufgrund des schönen Wetters ändern wi...

Tabelle 14.4: Der Variablenmodifikator truncate

`truncate` ergänzen Sie in der Template-Datei hinter dem Pipe-Zeichen `|`. Weitere Parameter geben Sie durch Doppelpunkt getrennt an: zuerst die gewünschte Zeichenlänge (der Standardwert ist 80), dann die Zeichen, die am Ende erscheinen sollen. Falls Sie als letzten Parameter `true` angeben, werden Wörter abgeschnitten.

Ein weiterer Variablenmodifikator ist `wordwrap`. Damit können Sie Text nach einer bestimmten Anzahl von Zeichen umbrechen.

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe
`$meinemeldung = "Aufgrund des schönen Wetters ändern wir unsere Pläne. Wir gehen ins Schwimmbad statt ins Kino";` `$smarty->assign("meldung", $meinemeldung);`	`{$meldung|wordwrap:40:" \n"}`	Siehe Abbildung 14.4

Tabelle 14.5: Der Variablenmodifikator wordwrap

Hinter den Modifikator `wordwrap` geben Sie nach einem Pipe-Zeichen die Zeichenanzahl an, nach der der Umbruch stattfinden soll und danach nach einem Doppelpunkt, über welche Zeichenkombination der Umbruch realisiert werden soll. Standardmäßig ist das `\n`, was sich in der Browseransicht nicht auswirkt. Deswegen ist hier `<br />\n` eine vernünftige Wahl.

Auch Bedingungen lassen sich in der Template-Datei einfügen. Zum Vergleichen dienen die aus PHP bekannten Operatoren wie ==, <, >, <=, >=. Wichtig ist, dass vor und nach den Operatoren Leerzeichen stehen. Sie sehen nun ein Beispiel für eine `if` – `elseif` – `else`-Verzweigung. Diese wird mit `{if}` eingeleitet und über `{/if}` beendet. Darin stehen `{elseif}` und `{else}` jeweils mit Vergleichsoperatoren.

smarty_oo_erweitert.php	beispiel_erweitert.tpl	Ausgabe

`$meinestadt = "München";`
`$smarty->assign("stadt", $meinestadt);`
	`{if $stadt == "München"}`
	`Viele Grüße in den Süden!`
	`{elseif $stadt == "Cuxhafen"}`
	`Schöne Zeit am Meer!`
	`{else}`
	`Viele Grüße`
	`{/if}`
		Viele Grüße in den Süden!

Tabelle 14.6: Eine bedingte Anweisung im Template

Tipp

Im Beispiel wurden die aus PHP bekannten Operatoren verwendet. Daneben gibt es aber auch `eq` anstelle von == und `lt` anstelle von <. Eine vollständige Auflistung finden Sie unter *http://www.smarty.net/manual/de/language.function.if.php*.

Die einzeln in diesem Abschnitt gezeigten Beispiele stehen gesamt jetzt noch einmal in den folgenden Dateien.

Hier sehen Sie die gesamte Beispieldatei *smarty_oo_erweitert.php*:

Listing 14.6: Die Beispiel-Smarty-Datei (smarty_oo_erweitert.php)

```
01 <?php
02 require_once "smarty_beispiel/meinsmarty.php";
03 $smarty = new MeinSmarty;
04 $meintitel = "Ein erster Test mit Smarty";
05 $meinestadt = "München";
06 $meinemeldung = "Aufgrund des schönen Wetters ändern wir unsere Pläne. Wir gehen
ins Schwimmbad statt ins Kino";
07 $meineliste = array("Sonne", "Mond", "Sterne");
08 $meinefarben = array("red"=>"rot", "green"=>"grün");
```

```
09 $smarty->assign("farbe", $meinefarben);
10 $bilder = array("blumen.jpg", "boot.jpg",
11                 "landschaft.jpg", "stadt_am_meer.jpg",
12                 "strand.jpg");
13 $max = count($bilder)-1;
14 $zufallszahl = rand(0, $max);
15 $meinbild = $bilder[$zufallszahl];
16 $smarty->assign("neuertitel", $meintitel);
17 $smarty->assign("stadt", $meinestadt);
18 $smarty->assign("meldung", $meinemeldung);
19 $smarty->assign("liste", $meineliste);
20 $smarty->assign("bild", $meinbild);
21 $smarty->display("beispiel_erweitert.tpl");
22 ?>
```

Und die gesamte zugehörige Template-Datei (Listing 14.7).

Listing 14.7: Die Template-Datei (beispiel_erweitert.tpl)

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04  <head>
05   <title>{$neuertitel}</title>
06   <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
07 </head>
08 <body>
09 Ich wohne in {$stadt}.<br />
10 <p>{$liste[0]}</p>
11 <p>Farbe: {$farbe.red}</p>
12 <h2>Beispiele für truncate</h2>
13 <p>{$meldung}</p>
14 <p>{$meldung|truncate:40:" ..."}</p>
15 <p>{$meldung|truncate:40:"...":true}</p>
16 <p><img src="bilder/{$bild}" /></p>
17 <h2>Beispiel für wordwrap</h2>
18 <p>{$meldung|wordwrap:40:"<br />\n"}</p>
19 <h2>Verzweigung</h2>
20 <p>
21 {if $stadt == "München"}
22   Viele Grüße in den Süden!
23 {elseif $stadt == "Cuxhafen"}
24   Schöne Zeit am Meer!
25 {else}
26   Viele Grüße
27 {/if}
28 </p>
29 </body>
30 </html>
```

Die Ausgabe zeigt Abbildung 14.4.

Abbildung 14.4: Ausgabe des Smarty-Beispiels

Hinweis

Das waren selbstverständlich noch nicht alle Möglichkeiten von Smarty. Wenn Sie mehr wissen wollen, finden Sie die gewünschten Informationen in der deutschsprachigen Dokumentation unter *http://www.smarty.net/manual/de/*.

Die Verwendung eines Template-Systems ist ein wesentlicher Schritt hin zu einer besseren Organisation des Codes. Oft sind hier aber noch weitere Trennungen notwendig, für die man heutzutage meist auf das MVC-Pattern setzt. MVC ist eine Abkürzung für die drei involvierten Komponenten: *Model – View – Controller*.

- Das Model (Modell) ist für die Daten und die Kommunikation mit der Datenbank zuständig.
- Der View (Präsentation) ist für die Darstellung verantwortlich.
- Die Controller-Schicht (Steuerung) verwaltet die Präsentation, reagiert auf Benutzeraktionen und entscheidet, welche Daten im Model geändert werden müssen.

Diese drei Komponenten arbeiten eng zusammen – sind aber in unterschiedlichen Dateien untergebracht. Das hat den Vorteil, dass spätere Änderungen und Erweiterungen leicht zu realisieren sind. Beispielsweise müssen Sie bei einer Umstellung der Datenvorhaltung nur Korrekturen am Model vornehmen, die anderen Schichten bleiben davon unbeeinflusst.

Das MVC-Pattern ist derzeit äußerst populär und es wird Ihnen an verschiedenen Stellen begegnen, wenn Sie beispielsweise ein PHP-Framework wie Zend, Symfony oder Ähnliche nutzen, oder wenn Sie tiefer in ein CMS wie Joomla! einsteigen.

Aber wohin auch immer Ihr weiterer Weg mit PHP Sie führen wird: Ich wünsche Ihnen viel Spaß und Erfolg!

Anhang

In Kapitel 2 haben Sie ein Beispiel gesehen, wie Sie PHP über die *php.ini*-Datei konfigurieren. Aber das ist nicht die einzige Möglichkeit zur Konfiguration von PHP. Was Sie an welchen Stellen einstellen können, finden Sie in diesem Anhang aufgeführt.

A.1 Konfigurationsmöglichkeiten für PHP

Konfigurationen an PHP können Sie an verschiedenen Stellen vornehmen:

- Die zentrale Stelle für die Konfiguration von PHP ist die *php.ini*-Datei. Hier können Sie alle Einstellungen vornehmen und diese gelten dann für alle PHP-Skripte – außer sie werden von Einstellungen an spezielleren Stellen überschrieben.
- Die *httpd.conf* ist die Hauptkonfigurationsdatei des Apache-Webservers. Auch hier können Einstellungen vorgenommen werden.
- Der Apache-Webserver kann auch über dezentral angelegte *.htaccess*-Dateien gesteuert werden. In diesen können Sie bestimmte PHP-Konfigurationen vornehmen. Diese gelten dann für die PHP-Skripte in dem Verzeichnis, in dem die *.htaccess*-Datei liegt.
- Außerdem können Sie bestimmte Einstellungen auch im Skript selbst vornehmen.

Tipp

Eine *.htaccess*-Datei ist eine Datei, die genauso – also *.htaccess* – heißt. Sie dient zur Konfiguration von Webservern wie Apache, wobei die dort vorgenommenen Einstellungen nur für das Verzeichnis gelten, in dem die *.htaccess*-Datei liegt. Damit Sie Einstellungen in der *.htaccess*-Datei setzen können, muss für die Verzeichnisse, in der die *.htaccess*-Dateien liegen, die Apache-Direktive `AllowOverride` auf `Options` oder `All` konfiguriert sein.

In *.htaccess*-Dateien können Sie wesentlich mehr als nur PHP-Konfigurationen ändern. Allgemeine Informationen zu diesen äußerst nützlichen Dateien lesen Sie unter *http://de.wikipedia.org/wiki/htaccess* und *http://de.selfhtml.org/servercgi/server/htaccess.htm*.

Nicht alle Einstellungen können an allen Stellen vorgenommen werden. Welche Direktiven Sie an welchen Stellen einstellen können, wird über mehrere Konstanten festgelegt. Diese finden Sie in der Übersicht *http://www.php.net/manual/de/ini.php* bei den einzelnen Direktiven aufgeführt:

Konstante	Bedeutung
`PHP_INI_USER`	Direktive kann im Skript selbst oder über die Windows Registry eingestellt werden.
`PHP_INI_PERDIR`	Direktive kann in *php.ini*, *.htaccess* oder *httpd.conf* gesetzt werden.
`PHP_INI_SYSTEM`	Direktive kann nur in *php.ini* oder *httpd.conf* gesetzt werden.
`PHP_INI_ALL`	Direktive kann überall gesetzt werden.

Tabelle A.1: Konstanten, die angeben, wo eine bestimmte Direktive gesetzt werden kann.

Achtung

Einstellungen über die *http.conf* und die *.htaccess* können Sie nur vornehmen, wenn PHP als Apache-Modul installiert ist.

A.1.1 Einstellungen in httpd.conf oder .htaccess setzen

Um Einstellungen in der *httpd.conf* oder der *.htaccess* zu setzen, gibt es vier Apache-Konfigurationsdirektiven.

- `php_value` *`Name Wert`*: setzt die Direktive auf den bei `Name` angegebenen Wert.

Handelt es sich beim Wert um einen Booleschen Wert, so benutzen Sie stattdessen:

- `php_flag` *`Name on/off`*: setzt die Direktive `Name` auf den Booleschen Wert `on` oder `off`

Nur in der *httpd.conf*, nicht jedoch in einer *.htaccess*-Datei, können Sie die beiden folgenden Konfigurationsdirektiven benutzen:

- `php_admin_value` *`Name Wert`*
- `php_admin_flag` *`Name Wert`*

Diese haben die Besonderheit, dass die durch sie gesetzten Werte nicht über eine *.htaccess*-Datei überschrieben werden können.

Eine *.htaccess*-Datei, in der Magic Quotes und Register Globals deaktiviert werden, sieht folgendermaßen aus:

```
php_flag register_globals off
php_flag magic_quotes_gpc off
```

Wenn Sie beispielsweise über eine *.htaccess*-Datei eine andere Einstellung angegeben haben, so sehen Sie das auch in der Ausgabe von `phpinfo()`. In dieser werden zwei Werte angezeigt, der lokal gesetzte (Local Value) und der allgemein geltende (Master Value). Sind beispielsweise über eine *.htaccess*-Datei die Magic Quotes ausgeschaltet, diese aber ansonsten aktiviert, sehen Sie in der Ausgabe eines in diesem Verzeichnis ausgeführten `phpinfo()`-Befehls als lokalen Wert `Off` und als Masterwert `On`.

magic_quotes_gpc	Off	On

Abbildung A.1: Zuerst wird der lokal gesetzte Wert und dann der globale angezeigt.

A.1.2 Informationen zur Konfiguration auslesen und Einstellungen im Skript setzen

Um Ihre Skripte an die aktuelle Konfiguration anzupassen, können Sie die aktuelle Konfiguration auslesen. Über `ini_get()` ermitteln Sie, wie eine bestimmte Direktive gesetzt ist, die Sie `ini_get()` übergeben:

```
echo "display_errors = " . ini_get("display_errors");
```

Tipp

Bei manchen Einstellungen existieren eigene `get`-Funktionen, beispielsweise zum Auslesen der Einstellung bzgl. der Magic Quotes: `get_magic_quotes_gpc()`.

Zum Setzen von Einstellungen dient `ini_set()`. Damit können Sie beispielsweise den Include-Path auch direkt im Skript selbst setzen. Mit dem folgenden Code wird zuerst der aktuelle Include-Path ermittelt und dann durch einen weiteren ergänzt:

Unix/Linux:

```
ini_set("include_path",ini_get("include_path").":/php/includes");
```

Windows:

```
ini_set("include_path",ini_get("include_path").";c:\\php\\includes");
```

Achtung

Es gibt einen wichtigen Unterschied bei der *php.ini*-Datei zwischen Linux und Windows: Pfade werden in der *php.ini* bei Linux mit einem Slash / angegeben, bei Windows mit einem Backslash \. Mehrere Pfadangaben wie beispielsweise bei den Include-Pfaden werden unter Linux mit Doppelpunkt, unter Windows mit Strichpunkt getrennt.

Stichwortverzeichnis

informit.de, Partner von Addison-Wesley, bietet aktuelles Fachwissen rund um die Uhr.

www.informit.de

In Zusammenarbeit mit den Top-Autoren von Addison-Wesley, absoluten Spezialisten ihres Fachgebiets, bieten wir Ihnen ständig hochinteressante, brandaktuelle deutsch- und englischsprachige Bücher, Softwareprodukte, Video-Trainings sowie eBooks.

wenn Sie mehr wissen wollen …

www.informit.de