

Matthias Kannengiesser

Know-how  
ist blau.

3. aktualisierte Auflage

# PHP 5 MySQL 5

Berücksichtigt  
PHP 5.3

- > PHP und MySQL verstehen und fehlerfrei anwenden
- > Dynamische Webseiten und komplexe Webanwendungen programmieren
- > Komplette PHP/MySQL-Umgebung XAMPP, Aptana Studio, MySQL-Referenz und alle Programmierbeispiele auf CD-ROM



FRANZIS

# Einleitung

PHP und MySQL sind ein unschlagbares Duo und dieses gilt es bei der Realisierung von dynamischen Webseiten im Griff zu haben. Das vorliegende Buch widmet sich ausführlich beiden beteiligten Protagonisten.

## **Über dieses Buch**

Die überarbeitete und aktualisierte Neuauflage des Bestsellers bietet Ihnen das Know-how aus zwei Büchern. Das Buch setzt sich aus folgenden Teilen zusammen:

- *Teil 1* – Der erste Teil des Buchs widmet sich der wohl populärsten Skriptsprache – PHP. Es soll Ihnen den Weg durch die mannigfaltige Struktur von PHP weisen. Durch die zahlreichen Codesnippets und Anwendungen sollen sowohl Umsteiger als auch fortgeschrittene Entwickler genügend Raum für eigene kreative Lösungsansätze erhalten. Darüber hinaus werden auch die grundlegenden Sprachelemente ausführlich beschrieben, sodass es zu einem unverzichtbaren Nachschlagewerk wird. Auf eines möchte ich jedoch bereits an dieser Stelle hinweisen: Dieser Teil des Buchs erhebt keinen Anspruch auf Vollständigkeit, da PHP in der fünften Generation bereits so umfangreich ist, dass man gut und gerne Dutzende von Büchern zum Thema verfassen könnte. Die zahlreichen PHP-Erweiterungen würden dieses Unterfangen zum Scheitern verurteilen.
- *Teil 2* – Der zweite Teil des Buchs befasst sich mit dem Wesen von Datenbanken. Den Schwerpunkt bildet dabei MySQL. MySQL gehört zu den populärsten Datenbanken und seit der fünften Generation auch zu den umfangreichsten. Ich verspreche Ihnen nicht, dass Sie innerhalb kürzester Zeit und nur nach der Lektüre dieses Buches zum MySQL-Spezialisten werden können. Erst die Arbeit an eigenen Projekten wird Ihnen zeigen, wie sinnvoll das Buch bei Ihrer alltäglichen Arbeit mit MySQL einsetzbar ist. Vorkenntnisse in MySQL sind sicher kein Grund, das Buch nicht zu lesen – im Gegenteil, auch für Fortgeschrittene sind weitergehende Informationen enthalten. Es handelt sich also nicht um eine oberflächliche Betrachtung, sondern vielmehr um eine Vertiefung des Stoffs.
- *Teil 3* – Der letzte Teil des Buchs liefert nützliche Informationen zu Sicherheitsaspekten.

**Tipp:** Die Buch-CD hält noch einige Überraschungen für Sie bereit. Ich habe keine Mühe gescheut, Ihnen einige nützliche Tools zusammenzutragen.

## **Up-to-date - PHP 5.3.0 und MySQL 5.1.36**

An dieser Stelle würde ich Sie gerne mit einigen Informationen zu den im Buch enthaltenen Skripten versorgen. Sämtliche Skripts sind sowohl unter PHP 5.2.10 als auch

PHP 5.3.0 getestet worden. Die datenbankbezogenen Anwendungen wurden allesamt unter MySQL 5.0.27 und 5.1.36 auf Herz und Nieren geprüft. Sie sollten bei Ihrer Arbeit mit PHP und MySQL möglichst auf aktuelle und stabile Releases zurückgreifen, wenn es darum geht, einen Produktionsserver zu betreiben.

Allein in PHP 5.3.0 wurden vom Entwicklerteam weit über 140 Bugfixes vorgenommen und diverse PHP-Erweiterungen erweitert und verbessert.

Beim Upgrade von PHP 5.x auf 5.3.0 ist bei einigen Erweiterungen auf die Unterschiede zu achten, daher empfiehlt es sich, einen Blick auf die folgende Datei zu werfen: <http://de3.php.net/manual/en/migration53.php>.

**Hinweis:** Die vollständige PHP 5.3.0 Changelog-Datei kann unter <http://php.net/ChangeLog-5.php> eingesehen werden.

### **Gute Voraussetzungen**

Sie sollten bereits einige HTML-Grundkenntnisse haben und sich mit dem Einsatz eines Webbrowsers auskennen. Grundlegende Programmierkenntnisse wären sicher auch nicht fehl am Platz. Aber besonders wichtig: Haben Sie Spaß beim Lesen, und nehmen Sie sich ausreichend Zeit zum Experimentieren.

### **Quelle – Website zum Buch**

Die Website zum Buch ist [www.atomicscript.de](http://www.atomicscript.de).

### **Der Autor**

*Matthias Kannengiesser* ist Dipl.-Informatiker und Projektmanager im IT-Bereich. Er arbeitet seit mehreren Jahren als IT-Consultant für namhafte Unternehmen. In den letzten Jahren hat er vor allem an der Entwicklung von PHP/MySQL-basierten Lösungen gearbeitet. Seit mehr als zehn Jahren hält er Seminare, Workshops und Vorträge zu den Themen ActionScript, Lingo, JavaScript, PHP und Datenbankdevelopment. Er ist bundesweit als Fachdozent für Institute und Unternehmen tätig und Autor für Magazine wie *Internet Intern*, *Internet World*, *MX Magazin* und *Internet Professionell*.

### **Danksagung**

Ich will mich von Herzen bei meinen lieben und geschätzten Freunden und Kollegen bedanken. Das sind insbesondere:

Caroline Kannengiesser – Dank an mein Schwesterherz für die Unterstützung

Dr. Peter Schisler, Ingrid Singer, Bianca Lange und Michael Wrobel (L4 Institut)

Team von [Apachefriends.org](http://Apachefriends.org) – Jungs, Ihr seid großartig und XAMPP ist unschlagbar!

Team von [Tutorials.de](http://Tutorials.de) – PHP und MySQL Forever

Team von [Traum-projekt.com](http://Traum-projekt.com) – Was kostet die Welt?

Alex, Bernd, Jens, Frank, Ina, Conni, Ralph, Christopher, Christian, Markus, Mario, Gökhan, Ralf und all diejenigen, die ich hier vergessen habe.

Einen besonderen Dank möchte ich den Herren Franz Graser und Peter Schmid-Meil aussprechen, meinen Lektoren beim Franzis Verlag: Danke für die kompetente Betreuung bei der Umsetzung dieses Buches.

Last but not least: Ganz viele Umarmungen und Küsse gehen an meine großartige und liebevolle Mama!

### **Feedback**

Ich würde mich über Reaktionen und Anregungen freuen. Sie erreichen mich unter folgender Adresse zu erreichen:

*matthiask@atomicscript.de*

Ihr

Matthias Kannengiesser



# Inhaltsverzeichnis

<b>Teil I – PHP</b> .....	<b>19</b>
<b>1 Internet-/Intranettechnologien</b> .....	<b>21</b>
1.1 CGI – Common Gateway Interface .....	21
1.2 Dynamische Webseiten .....	22
1.2.1 PHP als Skriptsprache .....	23
1.3 Wie arbeitet PHP? .....	23
1.4 PHP & HTML .....	25
1.5 Wie funktioniert eine Webanwendung? .....	26
1.6 Software für Webanwendungen .....	27
1.7 Datenbanken .....	31
1.7.1 Kompatibilität zu SQL-Standards .....	32
1.7.2 Eigenschaften von MySQL .....	33
1.7.3 MySQL-Anwendungsgebiete .....	34
<b>2 Installation und Konfiguration</b> .....	<b>37</b>
2.1 Vorbereitung .....	37
2.2 Installation unter Windows .....	38
2.2.1 WAMP .....	38
2.2.2 Apache konfigurieren .....	39
2.2.3 Installation von PHP unter Windows .....	40
2.3 Installation unter Linux .....	41
2.3.1 LAMP .....	41
2.3.2 Installation von PHP als CGI-Programm .....	41
2.3.3 Installation von PHP als Apache-Modul .....	41
2.4 Installations-Kits .....	42
2.4.1 XAMPP .....	42
2.4.2 Apache 2 Triad .....	49
2.4.3 WAMP5 .....	51
2.4.4 MAMP .....	52
2.4.5 Installations-Kits und Sicherheit .....	52
2.5 PHP-Konfiguration .....	53
2.5.1 Syntax der Init-Datei .....	53
2.5.2 Sprachoptionen .....	53
2.5.3 Leistungsbegrenzungen .....	54
2.5.4 Fehlerbehandlung und Protokollierung .....	54
2.5.5 Datenbehandlung .....	55
2.5.6 Pfade und Verzeichnisse .....	56
2.5.7 PHP-Erweiterungen für Windows .....	57

2.5.8	Moduleinstellungen .....	57
2.6	Sicherheit.....	59
2.6.1	Sicherheitsprobleme .....	59
2.6.2	Angriffsszenarien.....	60
2.7	Internet Service Provider und PHP.....	61
2.7.1	Zugangsdaten.....	61
2.7.2	Angebote von Providern .....	61
2.8	Überprüfen der Konfiguration .....	62
2.9	MySQL – Installation .....	64
2.9.1	Installation des MySQL-Datenbankservers .....	66
2.9.2	Installation auf Unix-/Linux-Systemen .....	66
2.9.3	Installation auf Windows-Systemen .....	69
2.9.4	Installation überprüfen .....	70
2.9.5	Kommandozeilenwerkzeuge von MySQL .....	71
2.9.6	Weitere Hilfsprogramme .....	73
2.9.7	Grafische MySQL-Clients.....	74
2.9.8	Anwendungen der MySQL-AB-Gruppe.....	84
2.9.9	Anwendung zur Datenbankmodellierung.....	89
2.10	Entwicklungsumgebungen.....	92
2.10.1	Entwicklungsumgebungen und Editoren.....	92
2.10.2	Zend Studio .....	93
2.10.3	Eclipse PDT .....	94
2.10.4	Aptana Studio PHP.....	94
2.10.5	NetBeans IDE.....	95
2.10.6	Adobe Dreamweaver CS3 .....	96
2.10.7	Macromedia Dreamweaver 8 .....	97
<b>3</b>	<b>Sprachelemente und Syntax.....</b>	<b>99</b>
3.1	Integration von PHP .....	99
3.1.1	Notationshinweise .....	100
3.1.2	Einbindung externer Skripts .....	100
3.2	Einführung in PHP.....	102
3.2.1	Ausdrücke .....	102
3.2.2	Anweisungen .....	105
3.2.3	Codezeile.....	106
3.2.4	Semikolons .....	106
3.2.5	Leerzeichen.....	108
3.2.6	Groß- und Kleinschreibung.....	109
3.2.7	Geschweifte Klammern .....	109
3.2.8	Runde Klammern .....	110
3.2.9	Schlüsselwörter .....	110
3.2.10	Zuweisungen .....	111
3.2.11	Echo-Befehl.....	112
3.2.12	Print-Befehl .....	112
3.2.13	Unterschied zwischen echo und print .....	113
3.2.14	Heredoc.....	113
3.2.15	Kommentare.....	115

3.3	Datentypen.....	117
3.3.1	Strings/Zeichenketten .....	117
3.3.2	Zahlen .....	122
3.3.3	Boolesche Werte .....	127
3.3.4	Objekte.....	130
3.3.5	Arrays .....	131
3.3.6	Resource Type.....	136
3.3.7	NULL .....	136
3.3.8	Besonderheiten der verschiedenen Typen .....	137
3.3.9	Typumwandlung .....	139
3.3.10	Datentypen bestimmen .....	142
3.4	Variablen .....	145
3.4.1	Was ist eine Variable?.....	145
3.4.2	Variablendefinition.....	146
3.4.3	L- und R-Wert .....	147
3.4.4	Benennen von Variablen .....	148
3.4.5	Variablenwerte.....	148
3.4.6	Umwandeln und Prüfen von Variablen .....	151
3.4.7	Gültigkeitsbereiche und Sichtbarkeit von Variablen .....	155
3.4.8	Dynamische Variablen.....	157
3.4.9	Vordefinierte Variablen.....	159
3.4.10	Einsatz von register_globals .....	161
3.5	Konstanten .....	165
3.5.1	Vordefinierte Konstanten.....	165
3.6	Operatoren.....	166
3.6.1	Operator-Rangfolge .....	167
3.6.2	Vorrang der Operatoren.....	168
3.6.3	Assoziativität der Operatoren .....	168
3.6.4	Arithmetische Operatoren.....	169
3.6.5	Zuweisungsoperator .....	170
3.6.6	Vergleichsoperatoren .....	173
3.6.7	Gleichheitsoperatoren.....	176
3.6.8	Logische Operatoren.....	178
3.6.9	Bit-Operatoren.....	180
3.6.10	String-Operator .....	184
3.6.11	Konditionaloperator .....	184
3.6.12	Gruppierungsoperator .....	185
3.6.13	Inkrement- bzw. Dekrementoperatoren .....	186
3.6.14	Objekterzeugungs-Operator .....	187
3.6.15	Array-Operatoren.....	188
3.6.16	Operatoren zur Programmausführung .....	189
3.6.17	Fehlerkontroll-Operatoren.....	189
3.7	Kontrollstrukturen .....	190
3.7.1	if-Anweisung.....	190
3.7.2	if-else-Anweisung.....	193
3.7.3	if-elseif-Anweisung .....	195
3.7.4	switch-case-Anweisung .....	197

3.7.5	while-Schleife.....	200
3.7.6	do-while-Schleife .....	202
3.7.7	for-schleife .....	203
3.7.8	foreach-Schleife .....	206
3.7.9	Verschachtelte Kontrollstrukturen.....	208
3.7.10	break .....	210
3.7.11	continue .....	210
3.8	Funktionen und Prozeduren.....	211
3.8.1	Funktionsargumente .....	212
3.8.2	Vorgabewerte für Parameter .....	214
3.8.3	Variable Argumentlisten .....	214
3.8.4	Rückgabewerte .....	216
3.8.5	Fehlercode als Rückgabewert .....	216
3.8.6	Dynamisch Funktionen erzeugen .....	217
3.8.7	Bedingte Funktionen.....	218
3.8.8	Verschachtelte Funktionen .....	218
3.8.9	Variablenfunktionen .....	220
3.8.10	Rekursive Funktionen .....	221
3.9	Referenzen in PHP.....	226
3.9.1	Was sind Referenzen?.....	226
3.9.2	Was leisten Referenzen? .....	226
3.9.3	Referenzen aufheben .....	228
3.9.4	Referenzen entdecken.....	229
3.10	Arrays .....	229
3.10.1	Was sind Arrays? .....	230
3.10.2	Terminologie .....	230
3.10.3	Arrays erzeugen .....	230
3.10.4	Arrays löschen.....	233
3.10.5	Mehrdimensionale Arrays.....	233
3.10.6	Arrayfunktionen .....	238
3.10.7	Funktionen für mehrere Elemente .....	245
3.10.8	Sortierfunktionen .....	250
3.10.9	Sonstige Arrayfunktionen .....	254
3.10.10	Nützliche Arrayfunktionen.....	257
3.10.11	Nützliche Arrayoperationen .....	278
3.11	Mathematische Funktionen.....	279
3.11.1	Umwandlungsfunktionen.....	279
3.11.2	Mathematische Konstanten.....	282
3.11.3	Zufallszahlen.....	282
3.12	Datums- und Zeitfunktionen .....	285
3.12.1	Kalenderfunktionen.....	285
3.12.2	Datumsfunktionen .....	287
3.12.3	Zeitfunktionen .....	291
3.13	Stringfunktionen .....	297
3.13.1	Ersetzen von Zeichen in Zeichenketten .....	297
3.13.2	Umwandeln, Teilen und Verbinden von Zeichenketten .....	299
3.13.3	Suchen und Vergleichen von Zeichenketten.....	302

3.13.4	Ausgabe von Zeichen und Zeichenketten.....	305
3.13.5	URL- und HTML-spezifische Zeichenkettenfunktionen .....	307
3.13.6	Zusätzliche Funktionen .....	311
<b>4</b>	<b>Programmierung mit PHP .....</b>	<b>317</b>
4.1	Formulare und PHP .....	317
4.1.1	GET und POST .....	317
4.1.2	Ermitteln von Formulardaten .....	319
4.1.3	Auswertung von Formularen .....	322
4.1.4	Formularelemente auf Existenz prüfen.....	325
4.1.5	Dynamische Formulare .....	326
4.1.6	Formulare über mehrere Seiten.....	329
4.1.7	Fragen zu Formularelementen.....	331
4.1.8	Prüfen auf fehlende oder fehlerhafte Eingaben .....	333
4.1.9	Formulardaten und globale Servervariablen .....	334
4.2	Daten via URL .....	336
4.2.1	Kodierung von Daten.....	337
4.2.2	Achtung – Escape-Zeichen .....	338
4.2.3	Arbeiten mit dem \$QUERY_STRING .....	338
4.2.4	Gleichlautende Variablen .....	340
4.3	Cookies via PHP .....	341
4.3.1	Spezifikation von Cookies.....	341
4.3.2	Cookies in PHP.....	343
4.3.3	Cookies löschen .....	349
4.4	Session-Management via PHP .....	349
4.4.1	Konfiguration des Session-Moduls.....	350
4.4.2	Session-Funktionen in PHP .....	352
4.4.3	Weitergabe der Session-ID über Cookies .....	356
4.4.4	Weitergabe der Session-ID über GET/POST .....	357
4.4.5	Weitergabe der Session-ID über header() .....	358
4.5	Überprüfung des Verbindungsstatus .....	358
4.6	Servervariablen .....	359
4.6.1	CGI-Umgebung .....	359
4.6.2	Erzeugen von eigenen Logfiles .....	360
4.7	Dateisystem via PHP.....	362
4.7.1	Informationen über Dateien und Verzeichnisse .....	362
4.7.2	Verzeichnisoperationen .....	365
4.7.3	Berechtigungen von Dateien und Verzeichnissen .....	366
4.7.4	Auslesen von Verzeichnissen .....	368
4.7.5	Dateioperationen und Dateifunktionen .....	370
4.7.6	Lesen und Schreiben von Dateien .....	372
4.7.7	Erzeugen und Schreiben von Dateien.....	377
4.7.8	Kopieren, Umbenennen und Löschen von Dateien.....	379
4.7.9	Serialisierung von Daten .....	380
4.7.10	Verriegelung von Dateien .....	381
4.7.11	Auslesen von CSV-Dateien .....	382
4.7.12	Nützliche Dateioperationen .....	383

4.7.13	Nützliche Verzeichnisoperationen .....	389
4.7.14	Datei-Upload via HTML-Formular .....	393
<b>5</b>	<b>Objektorientierte Programmierung mit PHP 5 .....</b>	<b>403</b>
5.1	Der Umgang mit Klassen und Objekten .....	403
5.1.1	Klassendefinition in PHP 5 .....	404
5.1.2	Objekte erzeugen und verwenden .....	405
5.1.3	Konstruktoren und Destruktoren .....	406
5.1.4	Zugriffsbeschränkung (Datenkapselung) .....	407
5.1.5	OOP – Rundgang.....	408
5.1.6	Objekte klonen .....	411
5.1.7	Klassenvererbung in PHP 5.....	415
5.1.8	Finale Klassen und Methoden .....	416
5.1.9	Abstraktion von Klassen und Methoden .....	416
5.1.10	Interface – Objektschnittstellen .....	419
5.1.11	Statische Eigenschaften und Methoden.....	421
5.1.12	Verweisoperator/Gültigkeitsbereichsoperator (::).....	423
5.1.13	Klassenkonstanten.....	425
5.1.14	Objekte – Referenzen und Kopien .....	426
5.1.15	Magische Methoden (Interzeptormethoden) .....	427
5.1.16	Typen-Hinweise (class type hints).....	432
5.1.17	Ausnahmebehandlung.....	434
5.1.18	Dereferenzierung von Objekten .....	439
5.1.19	Einsatz von instanceof.....	441
5.1.20	Neue Konstante __METHOD__ .....	441
5.1.21	Entwurfsmuster (Design Patterns) .....	443
5.1.22	Anpassung von PHP 4 auf PHP 5 .....	446
5.1.23	Praxis – Lebewesen .....	447
<b>6</b>	<b>Neuerungen in PHP 5.3 .....</b>	<b>451</b>
6.1	Syntax .....	451
6.1.1	Herdoc-Syntax .....	451
6.1.2	Nowdoc-Syntax .....	452
6.1.3	Lambda-Funktionen .....	452
6.1.4	Closures.....	454
6.2	Funktionen und Parameter.....	458
6.2.1	Funktionen.....	458
6.2.2	Parameter .....	462
6.2.3	Konstanten .....	463
6.3	Veraltet und wiederbelebt.....	464
6.3.1	Veraltet .....	464
6.3.2	Wiederbelebt .....	465
6.4	Objektorientierte Programmierung .....	465
6.4.1	Namespaces (Namensräume) .....	465
6.4.2	Late Static Bindings.....	471
6.4.3	Exkurs – Geschachtelte Exceptions .....	475
6.4.4	Exkurs – Funktionsobjekte .....	475

6.5	Datenbankspezifische Neuerungen .....	476
6.5.1	Sqlite3 .....	476
6.5.2	Client-/Server-Bibliothek mysqlnd .....	476
6.5.3	Standard PHP Library (SPL) .....	477
6.5.4	Garbage Collection .....	477
6.5.5	Erweiterungen intl und fileinfo .....	479
<b>7</b>	<b>Datenbankprogrammierung.....</b>	<b>481</b>
7.1	MySQL und PHP .....	481
7.1.1	MySQLi-Installation .....	482
7.1.2	MySQLi – erste Gehversuche .....	483
7.1.3	MySQLi und SQL-Abfragen .....	484
7.1.4	Referenz zur MySQLi-Unterstützung.....	487
7.1.5	Referenz zur MySQL-Unterstützung .....	490
7.2	Verwaltung von MySQL-Benutzern .....	495
7.2.1	Anlegen und Bearbeiten von Benutzern .....	496
7.3	PHP und MySQL-Verbindung .....	498
7.4	PHP und MySQL-Zugriffe .....	502
7.4.1	Datenbank erstellen .....	502
7.4.2	Datenbanktabelle erstellen .....	503
7.4.3	Datenverwaltung .....	506
7.4.4	Datenbanken und Tabellen löschen .....	519
7.4.5	Datenbanktabellen ändern.....	520
7.4.6	Verknüpfte Tabellen (WHERE/JOIN).....	523
7.4.7	Tabellen vereinigen (UNION).....	533
7.5	Backups von Daten und Tabellen .....	537
7.5.1	Sichern von Struktur und Daten .....	538
7.5.2	Ausführung von gespeicherten SQL-Befehlen .....	539
	<b>Teil II – MySQL .....</b>	<b>541</b>
<b>8</b>	<b>Installation .....</b>	<b>543</b>
8.1	MySQL-Server installieren .....	544
8.1.1	Installation auf Unix-/Linux-Systemen .....	544
8.1.2	Installation auf Windows-Systemen .....	547
8.1.3	Installation überprüfen .....	549
8.1.4	Die Konfigurationsdatei my.cnf / my.ini .....	550
8.1.5	Zeichensätze/Lokalisierung.....	551
8.1.6	MySQL-Installation aktualisieren.....	553
8.1.7	Mehrere MySQL-Server auf einem Rechner .....	553
8.1.8	LAMP (Linux-Apache-MySQL-PHP).....	555
8.1.9	WAMP (Windows-Apache-MySQL-PHP) .....	556
8.2	MySQL bei Internet-Service-Providern .....	557
8.2.1	Angebote.....	557
8.2.2	Funktionsumfang von MySQL feststellen.....	557
8.2.3	Server-Sharing (Webpace) .....	558
8.2.4	Eigener Webserver mit MySQL.....	559

8.3	Kommandozeilenwerkzeuge von MySQL .....	560
8.3.1	mysql – die SQL-Shell.....	560
8.3.2	mysqladmin .....	562
8.3.3	mysqlshow.....	562
8.3.4	Weitere Hilfsprogramme .....	562
<b>9</b>	<b>Datenbankentwurf.....</b>	<b>565</b>
9.1	Phasen der Datenbankentwicklung .....	565
9.2	Datenbankmanagementsysteme .....	566
9.3	Datenmodell.....	567
9.4	Datenbankentwurf mit ERM.....	568
9.5	Relationales Datenmodell.....	571
9.6	Primärschlüssel.....	572
9.7	Fremdschlüssel und referenzielle Integrität .....	572
9.8	Optimierung des Datenmodells .....	574
9.9	Implementierung und Nutzung von SQL .....	580
9.9.1	DCL-Befehle.....	580
9.9.2	DML-Befehle .....	580
9.9.3	DDL-Befehle.....	581
9.9.4	DQL-Befehle .....	581
<b>10</b>	<b>Datenbanken und Tabellen .....</b>	<b>583</b>
10.1	Datentypen.....	584
10.1.1	Zeichenketten-Datentypen.....	587
10.1.2	Numerische Datentypen .....	588
10.1.3	Datums- und Zeitdatentypen .....	589
10.1.4	Aufzählungen.....	590
10.1.5	Datentyp-Mapping .....	590
10.1.6	Datentypenübersicht .....	591
10.2	Datenbank anlegen und löschen.....	594
10.3	Tabellen.....	597
10.3.1	Tabellen anlegen.....	597
10.3.2	Schlüsselfelder für Tabellen .....	599
10.3.3	Indizes für Tabellen .....	600
10.3.4	Tabellentypen .....	602
10.3.5	Autowert definieren, Tabellen kopieren.....	611
10.3.6	Fremdschlüssel (FOREIGN KEY) .....	612
10.3.7	Ändern des Tabellenlayouts (ALTER TABLE) .....	613
10.3.8	Tabellen umbenennen und löschen .....	617
<b>11</b>	<b>Arbeiten mit Daten .....</b>	<b>619</b>
11.1	Benutzerwerkzeuge und -schnittstellen .....	619
11.2	Daten einfügen, ändern und löschen .....	620
11.2.1	Einfügen von Daten in Tabellen .....	620
11.2.2	Daten aktualisieren .....	623
11.2.3	Daten löschen.....	625
11.2.4	Daten aus anderen Datenbanken bzw. Programmen übernehmen.....	627

11.3	Befehle für die Datenausgabe.....	635
11.3.1	Abfragen mit SELECT .....	635
11.3.2	Vergleichsoperatoren .....	639
11.3.3	Abfragen mit logischen Operatoren .....	641
11.3.4	Mathematische Funktionen.....	642
11.3.5	Datums- und Zeitfunktionen .....	642
11.3.6	Zeichenketten.....	649
11.3.7	Auswahanweisungen.....	654
11.3.8	Zählen .....	656
11.3.9	Tabellen vereinigen (UNION).....	657
11.3.10	Verknüpfte Tabellen .....	658
11.3.11	Ausgabe sortieren .....	663
11.3.12	Deutsche Sortierung .....	664
11.3.13	Ausgabedatei über SELECT erzeugen .....	665
11.3.14	Abfragen analysieren.....	665
11.3.15	NULL-Marken .....	667
11.4	Unscharfe Suche .....	668
11.4.1	Suche mit LIKE und IN .....	668
11.4.2	Volltextsuche .....	669
11.4.3	Soundex .....	672
11.4.4	Reguläre Ausdrücke.....	673
11.5	Abfragen beschleunigen .....	676
11.5.1	Query Cache.....	676
11.6	Transaktionen .....	678
11.7	Benutzerdefinierte Funktionen .....	679
11.7.1	CREATE FUNCTION – Erzeugen von Funktionen .....	680
11.7.2	DROP FUNCTION – Löschen von Funktionen.....	681
11.8	Vorbereitete Anweisungen .....	681
<b>12</b>	<b>Verwaltung und Sicherheit.....</b>	<b>683</b>
12.1	Laufenden Betrieb überwachen.....	683
12.2	Protokollierung von Servervorgängen .....	685
12.2.1	Das Fehlerprotokoll .....	685
12.2.2	Laufende Betriebprotokollierung .....	685
12.3	Tabellenüberprüfung und -wartung.....	689
12.3.1	Tabellenüberprüfung .....	689
12.3.2	MyISAM-Tabellen reparieren .....	691
12.3.3	Tabellen optimieren .....	692
12.4	Sicherheit.....	694
12.4.1	Passwortsystem .....	694
12.4.2	Daten verschlüsselt speichern.....	703
12.4.3	SSH-Verbindungen.....	704
12.4.4	SSL-verschlüsselte Verbindungen .....	705
12.5	Backup und Datensicherung .....	709
12.5.1	Grundsätzliche Strategien für die Datensicherung.....	709
12.5.2	Backup mit mysqldump .....	711
12.5.3	Backup durch Datenbankkopie .....	714

12.5.4	Backup mit BACKUP TABLE .....	714
12.5.5	Datensicherung mit SELECT INTO OUTFILE.....	715
12.5.6	Replikationsmechanismen von MySQL .....	715
12.6	Uploads und Datensicherung bei Providern .....	717
12.7	Datenbanktests durchführen .....	719

**Teil III – Anhänge .....** 721

**A Sicherheit .....** 723

A.1	Schwachstellen und Gefahren .....	723
A.2	Webanwendungen und Sicherheit .....	725

**B CD-ROM und [www.buch.cd](http://www.buch.cd) .....** 727

B.1	Kapitel .....	727
B.2	Installations-Kits .....	727
B.3	PHP-Entwicklungsstudio .....	727
B.4	MySQL-Tools .....	727
B.5	Bonuskapitel .....	727
B.6	Zusätzlich auf <a href="http://www.buch.cd">www.buch.cd</a> .....	728

**Nachwort .....** 729

**Stichwortverzeichnis.....** 731

# 5 Objektorientierte Programmierung mit PHP 5

## 5.1 Der Umgang mit Klassen und Objekten

Die Zend Engine 2 ist der Versuch, PHP für den Einsatz im Enterprise-Sektor konkurrenzfähig zu machen und einzugliedern. Sicherlich wurde PHP auch schon vorher von Firmen gezielt eingesetzt. Die Anforderungen an eine Programmiersprache hinsichtlich Zuverlässigkeit und Sicherheit wurden jedoch keineswegs erschöpfend erfüllt.

Die wesentliche Änderung von PHP 5 und der von Zend entwickelten Zend Engine stellt die vollständig überarbeitete Unterstützung für die objektorientierte Programmierung in PHP dar. PHP 5 bietet gegenüber PHP 4 wesentlich bessere Möglichkeiten im Umgang mit Klassen und Objekten, wie z. B. Destruktoren, Exceptions (Fehlerbehandlungsroutinen), Interfaces (Schnittstellen), und eine verbesserte Kapselung. Im Gegensatz zu PHP 4 werden bei Zuweisungen und Übergaben Objekte immer als Referenz behandelt. In Zukunft dürfen Sie sich noch auf weitere Features wie Mehrfachvererbung und strikte Typisierung freuen.

**Achtung:** In PHP 4 werden Objekte bei Übergabe an Funktionen oder bei Zuweisungen immer als Kopie übergeben und nicht als Referenz!

Zunächst ein Blick auf die in PHP 5 hinzugekommenen Schlüsselwörter.

<i>Schlüsselwort</i>	<i>Beschreibung</i>
abstract	Deklariert Klassen oder Methoden als abstrakt.
clone	Wird verwendet, um eine Kopie eines Objekts zu erzeugen.
const	Definiert klassenspezifische Konstanten.
final	Deklariert Klassen oder Methoden als endgültig.
implements	Angabe, wenn eine Klasse ein Interface implementiert.
interface	Deklariert eine Objektschnittstelle.
instanceof	Entspricht <code>is_a()</code> und sollte stattdessen in PHP 5 verwendet werden. Gibt TRUE zurück, wenn das Objekt von einer Klasse abstammt.
private	Private Mitglieder wie Eigenschaften und Methoden sind für Aufrufer der Klasse nicht sichtbar.
protected	Geschützte Mitglieder wie Eigenschaften und Methoden sind für Aufrufer der Klasse nicht sichtbar, können jedoch in direkt abgeleiteten Klassen verwendet werden.

<i>Schlüsselwort</i>	<i>Beschreibung</i>
public	Öffentliche Mitglieder wie Eigenschaften und Methoden sind für alle Aufrufer der Klasse sichtbar. Dies ist der Standardwert, das heißt, ohne Angabe des Schlüsselworts sind alle Mitglieder öffentlich.
self	Kann in Klassen verwendet werden, um Bezug zu Methoden innerhalb derselben Klasse zu nehmen.
static	Methoden oder Eigenschaften können ohne Instanz eines Objekts verwendet werden.
try	Leitet einen Block ein, der der Ausnahmebehandlung unterliegt.
catch	Leitet einen Block ein, der eine spezifische Ausnahme behandelt.
throw	Generiert eine Ausnahme (engl. exception).
__autoload	Global verfügbare Funktion, mit deren Hilfe Klassenabhängigkeiten zur Laufzeit erfüllt werden können. Wird aufgerufen, wenn ein Objekt der Klasse erzeugt werden soll, die Klasse aber nicht deklariert ist.
__call	Ruft dynamisch Methoden auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__construct	Reservierter Name für den Konstruktor einer Klasse. Der Konstruktor wird aufgerufen, bevor das Objekt erzeugt wird. Dieser wird vor allem verwendet, um einen definierten Zustand zu erzeugen. Auslöser ist der Aufruf des Schlüsselworts <code>new</code> .
__destruct	Reservierter Name für den Destruktor einer Klasse. Der Destruktor wird aufgerufen unmittelbar bevor das Objekt zerstört wird. Dieser wird vor allem verwendet, um mit dem Objekt verbundene Ressourcen zu bereinigen.
__get	Ruft dynamisch Eigenschaften zum Lesen auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__set	Ruft dynamisch Eigenschaften zum Schreiben auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__METHOD__	Unechte Konstante. Wird diese innerhalb einer Klasse verwendet, enthält sie den Namen der Klasse und der Methode. Innerhalb einer Funktion, jedoch außerhalb einer Klasse enthält sie lediglich den Funktionsnamen.

### 5.1.1 Klassendefinition in PHP 5

Wenn es um die objektorientierte Programmierung geht, dann stellt die Definition einer Klasse den Ausgangspunkt dar. Um in PHP eine solche Klasse deklarieren zu können, wird das Schlüsselwort `class` benötigt, welches auch in diversen anderen Hochsprachen sein Unwesen treibt.

*Beispiel – Klasse (samt Mitgliedern, wie Eigenschaften und Methoden)*

```
<?php
// Klassen - Definition
class Produkt
{
```

```
// Klassen - Eigenschaften
public $name = "Vaio X20";
public $hersteller = "Sony";
public $preis = 1999.95;

// Klassen - Methode
public function Kaufen($kaeuffer)
{
    echo "Käufer: $kaeuffer\n";
    echo "Verkauf eines ";
    echo $this->name . " von ";
    echo $this->hersteller . "\n";
    echo "-----\n";
    echo "Preis: " . $this->preis;
}
}
?>
```

Definiert wurde hier eine Klasse mit dem Namen `Produkt`. Sie enthält die drei Eigenschaften `$name`, `$hersteller` und `$preis`, die mit Standardwerten belegt werden. Das Schlüsselwort `public` vor den Variablenamen deklariert sie als öffentliche Mitglieder. Zusätzlich enthält die Klasse noch eine Methode `Kaufen`, welche ebenfalls als öffentliches Mitglied deklariert wurde.

Bisher liegt lediglich die Klassendefinition vor, als Nächstes folgt die Erzeugung eines Objekts, um auf die Klassenbestandteile (Mitglieder bzw. Elemente) zugreifen zu können.

### 5.1.2 Objekte erzeugen und verwenden

Objekte sind in der Lage, Aktionen auszuführen. Dazu muss jedoch erst ein solches Objekt mithilfe des Schlüsselworts `new` erzeugt werden.

*Beispiel – Objekt erzeugen und auf Methode zugreifen*

```
// Objekt - Erzeugung
$einkauf = new Produkt();

// Methoden - Aufruf
echo "Onlineshop: \n";
$einkauf->Kaufen("Caroline");
```

*Ausgabe*

```
Onlineshop:
Käufer: Caroline
Verkauf eines Vaio X20 von Sony
-----
Preis: 1999.95
```

Das erzeugte Objekt `$einkauf` wird in einer Objektvariablen gespeichert. Anschließend kann das Objekt verwendet werden. Auf Eigenschaften und Methoden wird über die Verweissyntax `$objektname->...` zugegriffen.

### 5.1.3 Konstruktoren und Destruktoren

Der bereits aus PHP 4 bekannte Konstruktor ist Teil eines Phasenmodells, wobei das Objekt nach dem Initialisierungszustand (Konstruktor) zur Benutzung freisteht und am Ende beseitigt wird (Destruktor). Ein typisches Szenario, welches wir auch bei Aktionen im Alltag wieder finden, wenn wir beispielsweise etwas auf- und abbauen.

#### *Funktionsweise von Konstruktoren und Destruktoren*

Ein Konstruktor ist nichts anderes als eine Funktion, die immer dann aufgerufen wird, wenn das Objekt der Klasse instanziiert wird. Der Konstruktor wird ab sofort mit `__construct()` deklariert, obwohl er auch dann weiterhin als Konstruktor erkannt wird, wenn er den Namen der Klasse trägt. Konstruktoren und Destruktoren besitzen im Gegensatz zu gewöhnlichen Funktionen keinen Returntyp. Sämtliche Aufräumarbeiten, die zum sorgfältigen Beseitigen eines Objekts beitragen, übernimmt der parameterlose Destruktor `__destruct()`. Dazu gehört neben der zuletzt aufgerufenen Referenz die Freigabe jeglicher Speicherressourcen für das Objekt.

Ein Objekt existiert, wie alle anderen Variablen auch, bis zum Ende des Skripts oder bis es gezielt zerstört, also auf `NULL` gesetzt oder mit `unset()` vernichtet wird. Normalerweise ergeben sich daraus keine Konsequenzen, PHP sorgt automatisch für die Freigabe des belegten Speichers. Es gibt jedoch Anwendungsfälle, in denen externe Programme an der Kommunikation beteiligt sind, wie z. B. Datenbanken. In solchen Fällen wäre es fatal, wenn ein Objekt eine Verbindung zur Datenbank herstellt und dann zerstört wird, während die Verbindung bestehen bleibt. Es entstehen verwaiste Verbindungen. Verfügt eine Datenbank lediglich über rudimentäre Kontrollmechanismen oder eine begrenzte Anzahl von erlaubten Verbindungen, führt dies früher oder später zu Fehlern. Um das Verhalten am Ende der Existenz eines Objekts zu kontrollieren, sind somit Destruktoren genau das richtige Mittel. Sie werden unmittelbar vor der endgültigen Zerstörung aufgerufen.

#### *Beispiel – Konstruktor und Destruktor*

```
<?php
class MeineKlasse {
    public function __construct() {
        echo "Befinden uns im Konstruktor.<br>";
        $this->name = "MeineKlasse";
    }

    public function __destruct() {
        echo "Zerstöre die Klasse: " . $this->name . "<br>";
    }
}

$objekt = new MeineKlasse();

?>
```

#### *Ausgabe*

```
Befinden uns im Konstruktor.
Zerstöre die Klasse: MeineKlasse.
```

*Beispiel – ohne public-Schlüsselwort*

```
<?php
class Lebewesen
{
    function __construct()
    {
        echo "Konstruktor";
    }
    function __destruct()
    {
        echo "Destruktor";
    }
}

$objekt = new Lebewesen();
print_r($objekt);
?>
```

*Ausgabe*

```
Konstruktor Lebewesen Objekt
(
)
Destruktor
```

**Achtung:** Bei abgeleiteten Klassen werden Konstruktoren und Destruktoren der Elternklasse nicht automatisch ausgeführt und müssen explizit mit `parent::__construct()` beziehungsweise mit `parent::__destruct()` aufgerufen werden. Das Schlüsselwort `parent` verweist dabei auf die Klasse, von der mit `extends` geerbt wurde. Der Konstruktor bzw. Destruktor wird über seinen Namen aufgerufen. Nach `parent` darf lediglich mit dem statischen Verweisoperator `::` gearbeitet werden, da zum Zeitpunkt des Konstruktoraufrufs das Objekt noch nicht existiert und deshalb die Definition direkt benutzt wird.

### 5.1.4 Zugriffsbeschränkung (Datenkapselung)

Eigenschaften und Methoden von Klassen lassen sich mit den Schlüsselwörtern `private` und `protected` vor unerlaubtem Zugriff schützen. Damit ist es möglich, die Sichtbarkeit von Eigenschaften und Methoden einzuschränken. Zugriff auf als `private` deklarierte Eigenschaften und Methoden besteht nur innerhalb der Klasse selbst, Eigenschaften oder Methoden, die als `protected` deklariert wurden, stehen dagegen zusätzlich auch in abgeleiteten Klassen zur Verfügung. Das Schlüsselwort `public` schränkt den Zugriff in keiner Weise ein und entspricht dem Klassenmodell in PHP 4. Mit `public`, `private` oder `protected` kann in PHP 5 der Zugriff auf Eigenschaften, Methoden und Klassen genau festgelegt und eine gute Kapselung erreicht werden.

Zusammengefasst kann man sagen: Es handelt sich bei `public`, `private` oder `protected` um Schlüsselwörter, die sich zum Verstecken und Kapseln von Daten innerhalb von Klassen eignen und mit deren Hilfe man in der Lage ist, die Daten nur noch den in der

Klasse definierten Methoden zugänglich zu machen und somit den unbefugten Zugriff von außen strikt zu unterbinden.

### **Einsatzmöglichkeiten und Funktionsweise**

Mit der Datenkapselung wird die Trennung von Nutzungs- und Implementierungsschicht, von Realisierung und Nutzung verfolgt: Ein Entwickler, der eine Klasse eines anderen Entwicklers nutzen möchte, braucht die internen Abläufe der Klasse nicht zu kennen, er verwendet nur die vereinbarte Schnittstelle.

Zu diesem Zweck werden den Eigenschaften und Methoden einer Klasse bei ihrer Deklaration eine von drei möglichen Sichtbarkeiten zugewiesen:

- `public` – Standardwert. Objekte sämtlicher Klassen können die Eigenschaften oder die Methoden sehen und verwenden. Aus Gründen der Abwärtskompatibilität ist die Angabe optional und kann entfallen.
- `private` – Nur Objekte derselben Klasse können die Eigenschaften oder die Methoden sehen und verwenden. Sie sollten beachten, dass abgeleitete Klassen oder Aufrufe von Objekten nicht darauf zugreifen.
- `protected` – Verhält sich wie `private`, jedoch dürfen Objekte von Subklassen auf `protected`-deklarierte Eigenschaften und Methoden ihrer Superklasse zugreifen.

**Hinweis:** `public` stellt im Grunde nichts anderes dar als ein Alias von `var`, welches in PHP 4 eingesetzt wird, um Klassenattribute festzulegen. Die Deklaration von Eigenschaften mit `var` wird weiterhin unterstützt, sollte aber in PHP 5 nicht verwendet werden, es sei denn, die Skripts sollen PHP 4-kompatibel sein.

## 5.1.5 OOP – Rundgang

In älteren PHP-Generationen wurden Objekte wie simple Datentypen behandelt, dies hat sich nun geändert. Objekte verweisen nun nicht mehr auf ihren Wert (*by value*), sondern auf ihr Handle (*by reference*), ein Handle stellt dabei einen Verweis auf einen Objektbezeichner dar.

Um die Neuerungen aus PHP 5 im Überblick darzustellen, sollten wir uns einem Beispiel zuwenden.

### *Beispiel*

```
<?php

//--- class ---/
//
// Aufruf zum erstellen einer Klasse!
// Alles, was sich innerhalb der geschweiften
// Klammern des Klassenaufrufs befindet,
// unterliegt der strengen Klassenhierarchie
// von PHP 5

class MeineKlasse {
```

```

//
//--- var ---
//
// Die Variablendefinitionen, die innerhalb
// der Klasse existent sind! Optional können
// hier Parameter vordefiniert werden!
// (String,Array,Boolean,Integer)
//
var $zeit;
var $eingabe;
var $meldung = "Ausgabe beendet!";

//
//--- private ---
//
// Mit Private definiert man Parameter, die nur
// innerhalb dieser Klasse aufgerufen werden
// können und somit NICHT vererbbar sind und von
// sogenannten Subklassen (Unterklassen) nicht
// abgerufen werden können!
//
private $MathePI;

//
//--- __construct ---
//
// Diese Funktion wird beim Aufruf der Klasse
// Automatisch aufgerufen! Dem Konstruktor kann
// man optionale Parameter beim Aufruf der Klasse
// übergeben. Dieser dient dazu, die Variablen
// eines Objekts zu initialisieren. Er kann und
// darf NIE Funktionsergebnis liefern!
//
function __construct($eingabe, $zeit) {
    // Nehme den Parameter entgegen und weise ihm die
    // vordefinierten Variable eingabe und zeit zu.
    $this->eingabe = $eingabe;
    $this->zeit = $zeit;
}

//
//--- protected ---
//
// Mit protected werden innerhalb der Klasse
// Funktionen definiert, die nur in der Klasse
// selbst von anderen Funktionen aufgerufen
// werden können! Sie sind daher außerhalb
// der Klasse NICHT ausführbar.
//
// Hinweis: print & echo Befehle haben hier

```

```

// nichts zu suchen!
//

protected function saved_funk() {
    // Nehme Eigenschaft input und rufe die private
    // Funktion auf!
    return floor(($this->eingabe)*($this->private_funk()));
}

//
//--- public ---
//
// Mit public hingegen definiert man eine
// Funktion, die innerhalb als auch außerhalb
// der Klasse angesprochen werden darf! Dies
// gilt übrigens auch für Variablen!
//

public function ausgabe_funk() {
    // Rufe die geschützte Funktion auf!
    return $this->saved_funk();
}

//
//--- private ---
//
// Der Parameter private ist mit der protected-
// Funktion vergleichbar, wobei er noch strikter
// vorgeht. Er verhindert eine Vererbung an
// weitere Klassen. Privat ist nur in der
// definierten Superklasse erreichbar und sonst
// nirgends!
//

private function private_funk() {
    // Zuweisung des Werts PI an die Eigenschaft
    // MathePi
    return $this->MathePI = M_PI;
}

//
//--- __destruct ---
//
// Ist eine Callback-Funktion, die bei jedem
// Aufruf der Klasse zurückgegeben wird! Daher
// auch NICHT Explizit aufgerufen werden muss!
// Sie ist dann sinnvoll, wenn man einen bestimmten
// Wert fest zurückgeben möchte! Es werden nur
// echo- und print-Befehle wiedergegeben!
//
// Ausnahme ist hier das Schreiben in Sessions
// oder Cookies, die zuvor definiert wurden!
//

```

```

function __destruct() {
    $inhalt = "Ausgabe " . $this->zeit . "<br>";
    $inhalt .= "Von <b>" . __FUNCTION__ . "</b> aus " . $this;
    $inhalt .= " der Superklasse <b>" . __CLASS__ . "</b><br>";
    $inhalt .= $this->meldung . "<br>";
    echo $inhalt;
}
}
?>

```

Nachdem Sie sich mithilfe der Beispielklasse einen Überblick über die Struktur einer Klasse verschafft haben, gehen wir nun dazu über, ein Objekt mithilfe der Klasse zu erzeugen.

```

<?php
echo "<h3>MeineKlasse</h3>";

$zeitStempel = date("h:i:s ", mktime());

// Objekt erzeugen
$meinObjekt = new MeineKlasse(16, $zeitStempel);

// Ansprechen der public-Funktion
$ausgabe = $meinObjekt->ausgabe_funk();
echo "Ergebnis: " . $ausgabe . "<br>";

?>

```

### Ausgabe

```

MeineKlasse
Ergebnis: 50
Ausgabe 11:45:14
Von __destruct aus Object id #1 der Superklasse MeineKlasse
Ausgabe beendet!

```

Des Weiteren ist es möglich, ein Objekt zu klonen und ihm neue Parameter zu übergeben.

```

<?php

// Klon von meinObjekt
$neuesObjekt = clone $meinObjekt;

// Abruf der Funktion über den Klon
$klonausgabe = $neuesObjekt->ausgabe_funk();
echo "Ergebnis: " . $klonausgabe . "<br>";

?>

```

## 5.1.6 Objekte klonen

Das durch die Zend Engine 2 veränderte Objektreferenzverhalten führt dazu, dass man stets eine Referenz auf eine Ursprungsklasse erzeugen kann und niemals eine Kopie eines Objekts samt Eigenschaften erhält. Wenn man komplexe Datenstrukturen hat und eine Klasse A erzeugt, die im weiteren Verlauf noch benötigt wird, es aber zusätzlich

noch eine Klasse B gibt, die eine Kopie benötigt, um damit einzelne Teile der Objektstruktur zu bearbeiten, ohne dabei die Ursprungsinstanz der Klasse A zu verändern, spielt das neu eingeführte Klonen von Objekten eine bedeutsame Rolle. Hierdurch erhält man sämtliche Eigenschaften des Ursprungsobjekts, inklusive aller Referenzabhängigkeiten.

Wird nun eine Kopie eines Objekts benötigt, kann diese Kopie mithilfe von `clone` erzeugt werden. Ein Objekt wird nach dem Aufruf von `clone $objekt` geklont.

### *Beispiel*

```
<?php
class Fahrzeug
{
    function __construct($typ)
    {
        $this->name = $typ;
    }
    function ProduziereFahrzeug($obj, $name)
    {
        $obj->name = $name;
    }
}

$mobil = new Fahrzeug("PKW");
$neues_mobil = clone $mobil;

$neues_mobil->ProduziereFahrzeug($neues_mobil, "Fahrrad");

// Ausgabe - PKW
echo $mobil->name;
// Ausgabe - Fahrrad
echo $neues_mobil->name;

echo "Original Objekt:\n";
print_r($mobil);
echo "\n\n";
echo "Geklontes Objekt:\n";
print_r($neues_mobil);

?>
```

### *Ausgabe*

```
PKW
Fahrrad

Original Objekt:
Fahrzeug Object
(
    [name] => PKW
)

Geklontes Objekt:
Fahrzeug Object
(
    [name] => Fahrrad
)
```

Sollen sämtliche Objekteigenschaften geklont werden, dann wird zu allererst überprüft, ob Sie die `__clone()`-Methode selbstständig definiert haben, um das Klonen selbst zu übernehmen. In diesem Fall sind Sie dafür verantwortlich, die notwendigen Eigenschaften im erzeugten Objekt zu deklarieren. Sollte dies nicht der Fall sein, wird die interne Methode `__clone()`, über die sämtliche Klassen verfügen, für dieses Objekt ausgeführt, die sämtliche Objekteigenschaften dupliziert. Bei einer abgeleiteten Klasse kann die `__clone()`-Methode der Elternklasse mit `parent::__clone()` verwendet werden.

### Beispiel

```
<?php
class Lebewesen
{
    public $name = "";

    function __construct()
    {
        $this->name = "Zelle";
    }
    function __clone()
    {
        $this->name = "Klon-Zelle";
        echo "Ich wurde geklont: ";
    }
}

class Menschen extends Lebewesen
{
    function __construct()
    {
        parent::__clone();
    }
}

$objekt = new Menschen();
print_r($objekt);
?>
```

### Ausgabe

```
Ich wurde geklont: Menschen Object
(
    [name] => Klon-Zelle
)
```

### Beispiel – Vertiefung

```
<?php
class Form {
    public $form_farbe;
    public $form_breite;
    public $form_hoehe;

    function setze_farbe($farbe) {
        $this->form_farbe = $farbe;
    }
}
```

```

    }

    function __clone() {
        $this->form_farbe = $this->form_farbe;
        $this->form_breite = 100;
        $this->form_hoehe = $this->form_hoehe;
    }
}

$form_objekt = new Form();
$form_objekt->setze_farbe("blau");
$form_objekt->form_breite = 300;
$form_objekt->form_hoehe = 300;

$clone_objekt = clone $form_objekt;
echo $clone_objekt->form_farbe . "<br>";
echo $clone_objekt->form_breite . "<br>";
echo $clone_objekt->form_hoehe . "<br>";

?>

```

### *Ausgabe*

```

blau
100
300

```

Beim Beispiel handelt es sich um eine fiktive Form, wobei sich inmitten der Klasse die `__clone()`-Methode befindet. Dadurch besteht die Möglichkeit, Objekteigenschaften mit neuen Werten zu belegen. Nach der Instanz des Form-Objekts werden die Klassenvariablen mit einem Wert initialisiert und die Objekteigenschaften werden weiterführend durch den Aufruf von `clone $form_objekt` geklont. Bei den nachfolgenden Ausgaben des geklonten Objekts ist festzustellen, dass die Breite innerhalb der `__clone()`-Methode modifiziert wurde und daher einen veränderten Wert ausgibt. Der Farb- und Höhenwert wurde nicht verändert und entspricht dem der zuvor initialisierten Klassenvariablen.

### *Beispiel – abschließende Anwendung*

```

<?php

class Adresse {
    static $id = 0;

    function Adresse() {
        $this->id = self::$id++;
    }

    function __clone() {
        $this->id = self::$id++;
        $this->vorname = $this->vorname;
        $this->nachname = $this->nachname;
        $this->ort = "New York";
    }
}

$obj = new Adresse();
$obj->vorname = "Matthias";

```

```

$obj->nachname = "Kanengiesser";
$obj->ort = "Berlin";

print $obj->id . "<br>";

$clone_obj = clone $obj;
print $clone_obj->id . "<br>";
print $clone_obj->vorname . "<br>";
print $clone_obj->nachname . "<br>";
print $clone_obj->ort . "<br>";

?>

```

### Ausgabe

```

0
1
Matthias
Kanengiesser
New York

```

## 5.1.7 Klassenvererbung in PHP 5

An der Klassenvererbung wurden kaum Änderungen vorgenommen. Achten Sie bitte darauf, dass die als `private` deklarierten Methoden nicht an Subklassen oder vererbte Klassen weitergegeben werden, sie sind nicht vererblich.

```

<?php
//
// Superklasse erweitern
//

class ErweiterteKlasse extends MeineKlasse {
    var $neueingabe;

    function __construct($in) {
        $this->neueingabe = $in;
    }

    public function nehmeInfo() {
        $daten = "Rufe mit ErweiterteKlasse Funktionen in MeineKlasse auf
: ";
        $daten .= $this->ausgabe_funk() + $this->neueingabe . "<br>";
        $daten .= "Wie Sie sehen wird die private Methode nicht";
        $daten .= " zurückgegeben! ;) <br>";
        return $daten;
    }
}

$erweitertesObjekt = new ErweiterteKlasse(5, $zeitStempel);
echo $erweitertesObjekt->nehmeInfo();

?>

```

## Erweiterung von Subklassen

Natürlich sind Sie nicht nur auf eine Vererbungsebene beschränkt. Sie können eigentlich auch eine Subklasse oder eine Subklasse einer Subklasse erweitern. Zum Beispiel könnten Sie eine Klasse `Mercedes` schreiben, die die Klasse `Auto`, oder eine Klasse `Lear`, die die Klasse `Plane` erweitert. Der Ablauf ist immer genau so, als würden Sie eine ganz normale Klasse erweitern. Es ist vollkommen unwichtig, ob die Superklasse eine Subklasse einer anderen Klasse ist oder nicht.

### 5.1.8 Finale Klassen und Methoden

Sie haben bereits erfahren, dass sich Klassen mithilfe des Schlüsselworts `extends` vererben lassen. In manchen Fällen soll dies aber nicht so sein, entweder für eine Klasse als solche oder auch nur für einzelne Methoden. Denn manche Methoden sind für die Funktion der Objekte von elementarer Bedeutung. Gelingt der Schutz mit `private` nicht, da der Zugriff von außen benötigt wird, muss das Überschreiben durch das Schlüsselwort `final` verhindert werden. Von einer so gekennzeichneten Klasse kann nicht geerbt werden, und bei als `final` gekennzeichneten Methoden ist das Überschreiben verboten.

#### Beispiel – Syntax

```
// Finale Klasse
final class Produkt
{
    ...
    // Finale Methode
    final public function kaufen($kaeuffer)
    {
        ...
    }
}
```

**Hinweis:** Für Methoden kann `final` mit `private` und `protected` kombiniert werden; Eigenschaften können nicht `final` sein.

### 5.1.9 Abstraktion von Klassen und Methoden

Werden Klassen oder Methoden als `abstract` gekennzeichnet, wird der Benutzer explizit dazu aufgefordert, hier eigenen Code zu schreiben. Somit ist `abstract` das genaue Gegenteil von `final` – statt des ausdrücklichen Verbots folgt nun das ausdrückliche Gebot. Der Entwickler der Klassen gibt damit Struktur, Namen und Aufbau vor, nicht jedoch die konkrete Implementierung, da dies möglicherweise von der Anwendung abhängt.

Eine Ableitung von Objekten von abstrakten Klassen ist nicht möglich. Es muss deshalb immer eine Implementierung erfolgen. Dies gilt auch für abstrakte Methoden. Es ist jedoch möglich, eine Klasse als abstrakt zu definieren und einige der Methoden bereits voll auszuformulieren.

```

<?php
// Durch abstract wird die Klasse für direkte
// Aufrufe blockiert
abstract class SuperKlasse {
    // Die Variable $wert mit protected verriegeln
    protected $wert = 5;

    // Funktionsname zur Vererbung freigeben
    abstract function ausgabe();

    function multiplizieren($eingabe) {
        return $eingabe*$this->wert;
    }
}

// Unterklasse (Subklasse)
class ErweiterteKlasse extends SuperKlasse {
    // RICHTIG
    function ausgabe() {
        // Die Funktion ist der Abstract-Klasse bekannt und kann
        // somit auf sie zugreifen!
        return $this->multiplizieren(10);
    }

    // FALSCH
    function ausgeben() {
        // Dies kann nicht funktionieren, da die Funktion
        // ausgeben() der Abstract-KLASSE nicht bekannt ist!
        return $this->multiplizieren(10);
    }
}

// Aufruf der extends-Klasse
$testObjekt = new ErweiterteKlasse();

// Nicht vergessen! Die Funktion ausgabe() kann nur
// bei der extends(vererbten)-Klasse aufgerufen werden,
// nicht bei der Superklasse, die mit abstract
// verriegelt wurde!
echo $testObjekt->ausgabe();

?>

```

### *Ausgabe*

50

### *Beispiel*

```

<?php
// mit abstract die Klasse für direkte Aufrufe blocken
abstract class Fahrzeug {
    // Variable $tueren mit protected verriegeln
    protected $tueren = 4;

    // Gebe Methodenname zur Vererbung frei
    abstract function ausgabe();
}

```

```

function starten($wert) {
    return $wert . " mit " . $this->tueren . " Türen wurde gestartet!";
}
}

// Erstelle Subklasse und nehme mit extends die abstract-Oberklasse
class PKW extends Fahrzeug {
    // RICHTIG
    function ausgabe() {
        // Die Funktion ist der abstrakten Klasse bekannt und
        // kann somit auf sie zugreifen!
        return $this->starten("PKW");
    }

    // FALSCH
    function abfahren() {
        // Das kann nicht funktionieren, da die Funktion
        // abfahren() der abstrakten Klasse nicht bekannt
        // ist!
        return $this->starten("PKW");
    }
}

// Aufruf der Klasse
$meinkpw = new PKW();

// Achtung: Die Funktion ausgabe() kann lediglich bei der
// extends (vererbten) Klasse aufgerufen werden, nicht bei
// der Oberklasse, die mit abstract verriegelt ist!
echo $meinkpw->ausgabe();

?>

```

### Ausgabe

```
PKW mit 4 Türen wurde gestartet!
```

**Hinweis:** Der Sinn einer abstrakten Klasse liegt darin, dass wenn man eine Vererbung auf mehrere Oberklassen vornimmt, es zu keinen Variablenverletzungen kommen soll. Sie ist also bei mehrfacher Vererbung sehr zu empfehlen!

Abschließend noch einige Besonderheiten, die es beim Einsatz von `abstract` zu beachten gilt:

- Von abstrakten Klassen kann kein Objekt instanzliert werden.
- Von einer abstrakten Klasse kann nur abgeleitet werden.
- Methoden abstrakter Klassen, die selbst als `abstract` definiert sind, müssen bei einer Ableitung implementiert werden.
- Eine abstrakte Klasse kann Methoden enthalten, die nicht als `abstract` definiert sind. Sobald jedoch eine Methode als `abstract` definiert ist, muss auch die Klasse insgesamt `abstract` sein.

### 5.1.10 Interface – Objektschnittstellen

Vor allem bei umfangreichen PHP-Anwendungen kommt es äußerst selten vor, dass lediglich ein einzelner Entwickler daran arbeitet. Angenommen, Sie gehören zu einem Team von Entwicklern, bei dem jeder einzelne Entwickler an einem separaten Teil – d. h. einer anderen Klasse – einer umfangreicheren PHP-Anwendung arbeitet. Die meisten dieser Klassen stehen miteinander nicht in Beziehung. Dennoch müssen die verschiedenen Klassen miteinander kommunizieren können. Sie müssen also eine Schnittstelle oder ein Kommunikationsprotokoll definieren, das alle Klassen befolgen.

Eine Möglichkeit wäre, dass Sie eine Kommunikationsklasse erstellen, welche sämtliche Methoden definiert, und dann jede einzelne Klasse dieser übergeordneten Klasse erweitern oder von ihr erben lassen. Da die Anwendung jedoch aus unverwandten Klassen besteht, ist es nicht sinnvoll, sämtliche Klassen in eine gemeinsame Klassenhierarchie zu pressen. Die bessere Lösung ist das Erstellen einer Schnittstelle, in der die Methoden deklariert werden, die diese Klassen zur Kommunikation verwenden. Anschließend können Sie jede Klasse diese Methoden implementieren lassen, d. h. ihre jeweils eigenen Definitionen zur Verfügung stellen. Für eine erfolgreiche Programmierung sind in der Regel keine Schnittstellen erforderlich. Werden Schnittstellen sinnvoll eingesetzt, kann das Design Ihrer Anwendung effektiver, skalierbarer und leichter zu pflegen sein.

#### **Schnittstellen – Definition**

In der objektorientierten Programmierung sind Schnittstellen (engl. interfaces) mit Klassen vergleichbar, deren Methoden deklariert wurden, die aber sonst nichts anderes »tun«. Eine Schnittstelle setzt sich somit aus »leeren« Methoden zusammen. Eine andere Klasse kann die von der Schnittstelle deklarierten Methoden implementieren. Objektschnittstellen können auch als Sonderfälle von abstrakten Klassen gelten. Schnittstellen werden mit dem Schlüsselwort `interface`, gefolgt von einem Namen, deklariert und enthalten per Definition nur abstrakte Methoden. Auf die explizite Angabe von `abstract` bei Methoden kann verzichtet werden. Im Unterschied zu abstrakten Klassen werden Schnittstellen mit dem Schlüsselwort `implements` von einer Klasse implementiert.

#### **Schnittstellen erstellen**

Schnittstellen werden auf dieselbe Art und Weise erstellt wie Klassen. Schnittstellen deklarieren Sie mit dem Schlüsselwort `interface`. Darauf folgen der Name der Schnittstelle und dann die geschweiften Klammern, die den Körper der Schnittstelle definieren.

Innerhalb von Schnittstellen dürfen keine Eigenschaften enthalten sein, und von sämtlichen Methoden darf nur der »Kopf« geschrieben werden, direkt abgeschlossen mit einem Semikolon, statt der geschweiften Klammern.

Bei der Implementierung wird wie bei der Klassenvererbung vorgegangen, anstatt `extends` kommt jedoch das Schlüsselwort `implements` zum Einsatz.

```
<?php
interface einInterface {
    public function machWas();
```

```

}
interface anderesInterface {
    public function machWasAnderes();
}
class MeineKlasse implements einInterface, anderesInterface {
    public function machWas() {
        // ...
    }
    public function machWasAnderes() {
        // ...
    }
}
?>

```

Wie Sie sehen, kann eine Klasse eine beliebige Anzahl an Schnittstellen über das Schlüsselwort `implements` implementieren.

Da die Klasse `MeineKlasse` die Schnittstellen `einInterface` und `anderesInterface` implementiert, können Objekte dieser Klasse beispielsweise an Methoden übergeben werden, die als Parameter ein Objekt vom Typ `MeineKlasse`, `einInterface` oder `anderesInterface` erwarten, sämtliche dieser Typanforderungen kann ein solches Objekt erfüllen.

Abschließend habe ich noch ein vertiefendes praktisches Beispiel für Sie.

### *Beispiel*

```

<?php
interface Warenkorb
{
    function ArtikelPlatzieren($artikel);
    function ArtikelEntfernen($artikel);
}
class Onlineshop implements Warenkorb
{
    private $bestellung = array();
    private $auftrag;

    function ArtikelPlatzieren($artikel)
    {
        array_push($this->bestellung, $artikel);
    }

    function ArtikelEntfernen($artikel)
    {
        if (in_array($artikel, $this->bestellung)) {
            $raus = array_search($artikel,$this->bestellung);

            unset($this->bestellung[$raus]);
        }
    }
}

```

```

function Bestellen()
{
    foreach ($this->bestellung as $key)
    {
        $auftrag .= $key . "\n";
    }
    return $auftrag;
}

$kunde = new Onlineshop();
$kunde->ArtikelPlatzieren("Sony TV X100");
$kunde->ArtikelPlatzieren("Panasonic DVR");
$kunde->ArtikelPlatzieren("ActionScript Praxisbuch");
$kunde->ArtikelPlatzieren("5 Kilo Hanteln");

echo "Im Warenkorb (nach Artikelplatzierung):\n" . $kunde->Bestellen() .
"\n";

$kunde->ArtikelEntfernen("Panasonic DVR");

echo "Im Warenkorb (nach ArtikelEntfernen):\n" . $kunde->Bestellen() .
"\n";

?>

```

### Ausgabe

```

Im Warenkorb (nach Artikelplatzierung):
Sony TV X100
Panasonic DVR
ActionScript Praxisbuch
5 Kilo Hanteln

Im Warenkorb (nach ArtikelEntfernen):
Sony TV X100
ActionScript Praxisbuch
5 Kilo Hanteln

```

#### 5.1.11 Statische Eigenschaften und Methoden

Statische Eigenschaften und Methoden werden direkt von der Klasse aus angesprochen, und nicht über das Objekt. Damit teilen sich sämtliche Objekte einer Klasse diese Mitglieder, ganz gleich, ob es sich dabei um Eigenschaften oder Methoden handelt. Anstatt ein Objekt der Klasse zu erzeugen, um dann von diesem aus auf die Attribute und Methoden zugreifen zu können, kann immer direkt auf die Eigenschaften und Methoden der Klasse zugegriffen werden. Sollten Sie eine statische Eigenschaft oder Methode erzeugen wollen, verwenden Sie beim Deklarieren das Schlüsselwort `static`.

Einsatzfälle für statische Variablen und Methoden:

- Nutzen Sie statische Mitglieder für Aufgaben, die keinen Bezug zu den spezifischen Daten eines Objekts haben, beispielsweise Umrechnungen.
- Statische Mitglieder ermöglichen die Implementierung von Verweiszählern, wie beispielsweise Eigenschaften, die allen Objekten gleich sind.

- Statische Mitglieder ermöglichen den direkten Aufruf ohne vorherige Instanziierung. Dies ist sinnvoll, wenn man ohnehin nur ein Objekt benötigt.

```
<?php
class Klasse {
    static $static_var = 5;
    public $mein_prop = 'Hallo';

    public static function ausgabe() {
        return "Ein Text...";
    }
}

// Abfrage der statischen Variablen
echo Klasse::$static_var;

// Abfrage der statischen Funktion
echo Klasse::ausgabe();

$objekt = new Klasse;

// Abfrage der public-Variablen durch das Objekt
echo $objekt->mein_prop;

?>
```

### Beispiel – Anzahl der erzeugten Objekte bzw. Entwickler

```
<?php
class Entwickler
{
    static $zaehler;

    public function __construct()
    {
        Entwickler::$zaehler++;
    }

    public function GetEntwicklerAnzahl()
    {
        return Entwickler::$zaehler;
    }
}

$entwickler1 = new Entwickler();
$entwickler1 = new Entwickler();
$entwickler1 = new Entwickler();

echo "Es existieren " . $entwickler1->GetEntwicklerAnzahl() . "
Entwickler.";

?>
```

### Ausgabe

Es existieren 3 Entwickler.

Um auf einfache Weise eine Klasse zu schaffen, die in der Lage ist, die Anzahl ihrer erzeugten Objekte zu erfassen, ist das Schlüsselwort `static` Gold wert. Durch Einsatz von `static` ist die Variable `$counter` der Klasse `Entwickler` in den Klassenkontext

überführt worden. Diese Variable existiert für sämtliche Objekte lediglich einmal. Sobald ein neues Objekt erzeugt wird, wird der Wert im Konstruktor um 1 erhöht. Um dabei die Variablen `$counter` verarbeiten zu können, erfolgt der Zugriff über einen Klassenverweis, die Pseudovariablen `$this` kann nicht verwendet werden. Dabei wird der Zugriff durch den statischen Verweisoperator `::` und den vollständigen Namen der Variablen, inklusive `-$`-Zeichen, ermöglicht. Dies gilt sowohl für Zugriffe innerhalb der Klasse, wie im vorliegenden Beispiel, als auch für externe.

**Hinweis:** Statische Variablen und Methoden können darüber hinaus als `public`, `private` oder `protected` gekennzeichnet werden.

### Beispiel – statische Methoden

```
<?php
class Datenbank
{
    // Variablendeklaration
    protected $db;

    // Statische Methode
    public static function verbindeDB($host, $user, $password)
    {
        @$db = new mysqli($host, $user, $password);

        if (mysqli_connect_errno()) {
            printf("Verbindungsfehler: %s\n", mysqli_connect_error());
            exit();
        }
        return $db;
    }
    public function oeffneDB()
    {
        $this->db = $this->verbindeDB("localhost", "root", "");
        return $this->db;
    }
}
// Methodenaufruf ohne Objekterzeugung
// Lediglich bei static-Methoden möglich!
$privatzugang_a = Datenbank::verbindeDB("localhost", "root", "");
echo $privatzugang_a;

// Aus einem Objekt heraus
$zugang = new Datenbank();
$privatzugang_b = $zugang->oeffneDB();
echo $privatzugang_b;
?>
```

## 5.1.12 Verweisoperator/Gültigkeitsbereichsoperator (::)

Der Gültigkeitsbereichsoperator, auch Verweisoperator genannt, ist ein Kürzel, das Zugriff auf statische, konstante und überschriebene Mitglieder einer Klasse ermöglicht.

**Hinweis:** Wenn solche Elemente außerhalb der Klassendefinition angesprochen werden sollen, muss der Name der Klasse verwendet werden.

#### *Beispiel – :: außerhalb der Klassendefinition*

```
<?php
class Flugzeug
{
    const MAX_FLUGHOEHE = 15000;
}
echo Flugzeug::MAX_FLUGHOEHE . " Fuss";
?>
```

#### *Ausgabe*

```
15000 Fuss
```

### **Schlüsselwörter self und parent**

Die zwei speziellen Schlüsselwörter `self` und `parent` werden verwendet, um auf Mitglieder wie Eigenschaften und Methoden innerhalb einer Klassendefinition zuzugreifen. Die beiden Schlüsselwörter weisen folgende Besonderheiten auf:

- `self` – Kann in Klassen verwendet werden, um Bezug auf Methoden innerhalb derselben Klasse zu nehmen.
- `parent` – Verweist auf die Klasse, von der mit `extends` geerbt wurde.

#### *Beispiel – :: innerhalb der Klassendefinition*

```
<?php
class Flugzeug
{
    const MAX_FLUGHOEHE = 15000;
}
class A320 extends Flugzeug
{
    public static $hoehen_toleranz = 3000;
    public static function fliegen() {
        echo "Flughöhe (max): " . parent::MAX_FLUGHOEHE . " Fuss\n";
        echo "Flughöhen (Toleranz): " . self::$hoehen_toleranz . "
Fuss\n";
    }
}
A320::fliegen();
?>
```

#### *Ausgabe*

```
Flughöhe (max): 15000 Fuss
Flughöhen (Toleranz): 3000 Fuss
```

Wenn eine abgeleitete Klasse die Definition der Methode einer Mutter überschreibt, wird PHP die Methode ihrer Mutter nicht aufrufen. Es obliegt der abgeleiteten Klasse, ob die Methode der Mutterklasse aufgerufen wird oder nicht. Dies gilt ebenfalls für Konstruktoren und Destruktoren, Überladung und magische Methodendefinitionen.

#### Beispiel – Methode einer übergeordneten Klasse aufrufen

```
<?php
class Lebewesen
{
    protected function erzeugen() {
        echo "Lebewesen::erzeugen()\n";
    }
}

class Menschen extends Lebewesen
{
    // Definition der übergeordneten Klasse überschreiben
    public function erzeugen()
    {
        // Trotzdem die Funktion der übergeordneten Klasse aufrufen
        parent::erzeugen();
        echo "Menschen::erzeugen()\n";
    }
}

$mensch = new Menschen();
$mensch->erzeugen();
?>
```

#### Ausgabe

```
Lebewesen::erzeugen()
Menschen::erzeugen()
```

### 5.1.13 Klassenkonstanten

Vor PHP 5 waren Konstanten immer global, was den Einsatz etwas problematisch machte, da Namenskonflikte fast schon vorprogrammiert waren. Da sich Konstanten von Objekt zu Objekt nicht ändern, verhalten sie sich wie statische Mitglieder und werden auch genau wie diese verwendet. Der einzige Unterschied besteht darin, dass sich der Inhalt nicht verändern lässt – außer während der Definition. Diese Definition erfolgt innerhalb der Klasse.

Im Gegensatz zu den globalen Konstanten, die mit der `define()`-Funktion erzeugt werden, erfolgt die Deklaration in Klassen mit dem Schlüsselwort `const`.

Der Zugriff auf eine solche Konstante erfolgt über `Klassenname::Konstante`. Soll der Zugriff auf die Konstante aus einer Methode derselben Klasse erfolgen, so kann auch `self::Konstante` verwendet werden.

#### Beispiel – `const` und `static`

```
<?php
class MeineKlasse {
```

```

const Konstante = 16;
static $statisch = 1;

function erhoehen() {
    return self::$statisch++;
}
}

$a = new MeineKlasse;
$b = new MeineKlasse;

echo "MeineKlasse::Konstante = " . MeineKlasse::Konstante . "<br>";
echo "\$a->erhoehen() = " . $a->erhoehen() . "<br>";
echo "\$b->erhoehen() = " . $b->erhoehen() . "<br>";

?>

```

### Ausgabe

```

MeineKlasse::Konstante = 16
$a->erhoehen() = 1
$b->erhoehen() = 2

```

### Beispiel – Physik

```

<?php

class Physik
{
    const ATOM_GEWICHT = 1.00895;
}

echo "Wert: " . Physik::ATOM_GEWICHT;

?>

```

### Ausgabe

```

Wert: 1.00895

```

## 5.1.14 Objekte – Referenzen und Kopien

Das Sprachverhalten der Zend Engine sah es bisher vor, beim Anlegen oder bei der Übergabe von Objektinstanzen nicht das eigentliche Objekt zu übergeben, sondern lediglich eine Kopie davon, so wie man es von Variablen kennt. Dieses Verhalten wird als *copy by value* bezeichnet und sorgte oft für Verwirrung. Die eigentliche Problematik liegt darin, dass etwaige Änderungen der Eigenschaften des Objekts sich nicht auf das Objekt selbst auswirken würden, sondern auf die Kopie des Objekts. Um tatsächlich nur eine Referenz auf das Objekt anzuwenden, konnte man sich nur mit der expliziten Deklaration des `&new`-Operators behelfen.

Die Verhaltensweise und Umgangsweise von Objekten orientiert sich strikt an Hochsprachen wie Java. Dort wird eine Objektinstanz mit einem *object handle* referenziert (*by reference*) und dieses Handle bezieht sich in allen Aktionen auf das Ursprungsobjekt. Der große Vorteil dabei ist, dass PHP 5 dies implizit erledigt, was nicht nur eleganter und flexibler ist, sondern auch spürbar die Performance steigert. Der konsequente Wechsel zum *object by reference*-Paradigma bietet aber auch Vorteile beim

Umgang in puncto Usability und Semantik, wodurch Features wie Destruktoren und Dereferenzierung profitieren.

```
<?php
class Fahrzeug
{
    function __construct($typ)
    {
        $this->name = $typ;
    }
    function ProduziereFahrzeug($obj, $name)
    {
        $obj->name = $name;
    }
}

$mobil = new Fahrzeug("PKW");
$mobil->ProduziereFahrzeug($mobil, "LKW");
// Ausgabe - LKW
echo $mobil->name;
?>
```

### 5.1.15 Magische Methoden (Interzeptormethoden)

Die Interzeptormethoden oder auch magischen Methoden von PHP 5 werden automatisch beim Zugriff auf nicht bekannte Eigenschaften und Methoden eines Objekts, beim Versuch, ein Objekt einer nicht deklarierten Klasse zu erzeugen, sowie bei der Typumwandlung eines Objekts in einen String aufgerufen.

- `__autoload($className)` wird aufgerufen, wenn ein Objekt der Klasse `$className` erzeugt werden soll, die Klasse aber nicht deklariert ist.
- `__call($methodName, $parameters)` wird aufgerufen, wenn eine nicht deklarierte Methode `$methodName` mit einem Objekt aufgerufen wird. Der zweite Parameter `$parameters` enthält die Parameter des Methodenaufrufs.
- `__get($memberName)` wird aufgerufen, wenn lesend auf das Attribut `$memberName` eines Objekts zugegriffen wird, das Attribut aber nicht gesetzt ist.
- `__set($memberName, $value)` wird aufgerufen, wenn schreibend auf das Attribut `$memberName` eines Objekts zugegriffen wird und das Attribut vorher nicht gesetzt war. Der zweite Parameter `$value` enthält den Wert, mit dem das Attribut belegt werden soll.
- `__toString()` wird aufgerufen, wenn eine Typumwandlung eines Objekts in einen String durchgeführt werden soll.

#### **Einsatz von `__autoload()`**

Die global verfügbare `__autoload()`-Funktion kann verwendet werden, um eigentlich nicht definierte Klassen nachzuladen. Wird auf eine nicht definierte Klasse zugegriffen, wird, falls vorhanden, `__autoload()` aufgerufen und ausgeführt. Ist `__autoload()` ent-

sprechend implementiert, können Klassen bequem nachgeladen werden. Damit ist es möglich, Klassen zur Laufzeit erst einzubinden, wenn sie tatsächlich benötigt werden.

Das folgende Beispiel zeigt den einfachsten Fall der Verwendung von `__autoload()` und geht davon aus, dass alle Klassen in einer Datei deklariert sind, deren Name sich aus dem Namen der Klasse und der Dateiendung `.php` zusammensetzt.

#### *Beispiel – einfacher Klassenlader mithilfe von `__autoload()`*

```
<?php
function __autoload($klassenname)
{
    include_once("$klassenname.php");
}
$daten = new Kontakte($host, $nutzer, $passwort);
?>
```

In diesem Beispiel steht die Klasse `Kontakte` eigentlich nicht zur Verfügung, da sie weder definiert noch eingebunden ist. Beim Versuch, `Kontakte` zu instanziiieren, wird `__autoload()` aufgerufen und die Klasse mit `include_once()` nachgeladen.

**Hinweis:** Ist die geforderte Klasse nach Ausführung der `__autoload()`-Funktion weiterhin unbekannt, so wird eine Fehlermeldung ausgegeben.

#### *Einsatz von `__call()`*

Die `__call()`-Funktion wird aufgerufen, wenn versucht wird, eine nicht deklarierte Methode mit einem Objekt aufzurufen. Mit der `__call()`-Methode ist es möglich, dem Funktionsaufruf Parameter an die `__call()`-Funktion zu übergeben. Der `__call()`-Funktion stehen dabei zwei Parameter zur Verfügung:

- Im ersten steht der Name der aufrufenden Methode zur Verfügung.
- Im zweiten Parameter stehen die übergebenen Werte als numerisches Array zur Verfügung.

**Hinweis:** Die `__call()`-Methode wird zum Überladen von Methoden verwendet. Klassenmethoden können überladen werden, um eigenen in Ihrer Klasse definierten Code auszuführen, indem man diese speziell benannte Methode definiert.

#### *Beispiel*

```
<?php
class Handies
{
    public $anzahl = 0;

    function __call($funktionsname, $parameter)
    {
        $this->anzahl = count($parameter);
        echo "Aufruf von $funktionsname mit $this->anzahl Parameter \n";
        if ($this->anzahl > 0) print_r($parameter);
    }
}
```

```

}
$test = new Handies();
$test->SetzeHersteller("Nokia","Siemens");
$test->SetzePreise(99.95, 199.99, 50);
?>

```

### Ausgabe

```

Aufruf von SetzeHersteller mit 2 Parameter
Array
(
    [0] => Nokia
    [1] => Siemens
)
Aufruf von SetzePreise mit 3 Parameter
Array
(
    [0] => 99.95
    [1] => 199.99
    [2] => 50
)

```

Im folgenden Beispiel wird der Datentyp geprüft und anschließend an die passende Funktion übergeben!

```

<?php
// Datentyp prüfen via __call()
class Auswertung {
    function __call($eingabe,$inhalt) {
        if($eingabe=='pruefen') {
            if(is_integer($inhalt[0]))
                $this->ausgabe_integer($inhalt[0]);
            if(is_string($inhalt[0]))
                $this->ausgabe_string($inhalt[0]);
            if(is_array($inhalt[0]))
                $this->ausgabe_array($inhalt[0]);
        }
    }
    private function ausgabe_integer($daten) {
        echo("Der Wert " . $daten . " ist ein Integer!<br>");
    }
    private function ausgabe_string($daten) {
        echo("Der Wert " . $daten . " ist ein String!<br>");
    }
    private function ausgabe_array($daten) {
        echo("Die Werte " . implode(", ", $daten) . " sind in einem
Array!<br>");
    }
}

```

```

}
// Klassenaufruf
$test = new Auswertung();
$test->pruefen(3);
$test->pruefen("3");
$array = array(10,20,30);
$test->pruefen($array);
?>

```

### Ausgabe

```

Der Wert 3 ist ein Integer!
Der Wert 3 ist ein String!
Die Werte 10,20,30 sind in einem Array!

```

### Einsatz von `__set()` und `__get()`

Eine weitere Variante der `__call()`-Methode sind `__set()` und `__get()`. Mit ihnen kann man direkt beim Aufruf die Werte beeinflussen. Die beiden Methoden sind spezielle Methoden, auf die von außerhalb der Klasse als Attribute zugegriffen werden kann, welche jedoch in der Klasse selbst als Methoden definiert vorliegen. Einer der wichtigsten Vorteile der Methoden ist, dass sie Eigenschaften erzeugen können, die von außerhalb wie Attribute erscheinen, die intern jedoch mit komplexen Abläufen arbeiten.

### Funktionsweise von `__set()` und `__get()`

Wird auf Eigenschaften eines Objekts zugegriffen, die nicht explizit definiert sind, wird die `__set()`-Methode aufgerufen, um einen Wert zu definieren. Soll dieser Wert abgefragt werden, wird die `__get()`-Methode aufgerufen. Sind weder `__set()` noch `__get()` implementiert, kommt es bei einem Zugriff auf nicht definierte Eigenschaften zu Fehlern.

Eine Besonderheit stellt folgendes Verhalten der beiden Methoden dar:

- Wenn man mit `__set()` eine Eigenschaft definiert und mit `__get()` nicht, erhält man eine »Nur-Schreib«-Eigenschaft.
- Wenn man mit `__get()` eine Eigenschaft definiert und mit `__set()` nicht, erhält man eine »Nur-Lese«-Eigenschaft.

**Hinweis:** Die `__get()/__set()`-Methoden werden zum Überladen von Mitgliedern verwendet. Klassenmitglieder können überladen werden, um eigenen in ihrer Klasse definierten Code auszuführen, indem man diese speziell benannten Methoden definiert.

### Beispiel

```

<?php
class Handies
{
    private $mobils = array(
        "Nokia" => 10,
        "Siemens" => 20
    );
}

```

```

);

public function __get($varname)
{
    return $this->mobils[$varname];
}

public function __set($varname, $wert)
{
    $this->mobils[$varname] = $wert;
}
}

$handy = new Handies();
$handy->Nokia++;
$handy->Siemens++;

echo "Wert von Nokia: " . $handy->Nokia . "\n";
echo "Wert von Siemens: " . $handy->Siemens . "\n";

?>

```

### Ausgabe

```

Wert von Nokia: 11
Wert von Siemens: 21

```

### Einsatz von `__toString()`

Die neue `__toString()` Methode ermöglicht es Ihnen, eine Typumwandlung vorzunehmen und ein Objekt in einen String umzuwandeln.

Verfügt eine Klasse in PHP 5 über eine `__toString()`-Methode, so wird sie aufgerufen, wenn ein Objekt der Klasse in einen String umgewandelt werden soll. So gibt das folgende Beispiel »Der Kontostand beträgt 9999.95 Euro.« aus anstatt `Object id #1`. Letzteres wäre der Fall, wenn die Klasse `Bankkonto` über keine `__toString()`-Methodendeklaration verfügen würde.

```

<?php

class Bankkonto {
    private $guthaben = 9999.95;

    public function __toString() {
        return sprintf('Der Kontostand beträgt %01.2f Euro.', $this->guthaben);
    }
}

$meinKonto = new Bankkonto;
print $meinKonto;

?>

```

### Ausgabe

```

Der Kontostand beträgt 9999.95 Euro.

```

# Stichwortverzeichnis

- (Dekrement) 186
  - (Subtraktion) 169
  - (Vorzeichen) 169
  - ' Einfache Anführungszeichen 119
  - ! (Logisches Nicht) 179
  - != (Ungleichheit) 176
  - !== (Strikte Ungleichheit) 177
  - \$\_SERVER[QUERY\_STRING] 338
  - \$AUTH\_TYPE 359
  - \$CONTENT\_LENGTH 359
  - \$CONTENT\_TYPE 359
  - \$GATEWAY\_INTERFACE 360
  - \$GLOBALS 156
  - \$HTTP\_ACCEPT 360
  - \$HTTP\_COOKIE 360
  - \$HTTP\_REFERER 360
  - \$HTTP\_USER\_AGENT 360
  - \$PATH\_INFO 360
  - \$PATH\_TRANSLATED 360
  - \$QUERY\_STRING 360
  - \$REMOTE\_ADDR 360
  - \$REMOTE\_HOST 360
  - \$REMOTE\_IDENT 360
  - \$REMOTE\_METHOD 360
  - \$REMOTE\_USER 360
  - \$\_SCRIPT\_NAME 360
  - \$SERVER\_NAME 360
  - \$SERVER\_PORT 360
  - \$SERVER\_PROTOCOL 360
  - \$SERVER\_SOFTWARE 360
  - % (Modulo) 170
  - && (Logisches Und) 178
  - () 110
  - () (Gruppierungsoperator) 185
  - \* (Multiplikation) 170
  - / (Division) 170
  - :: 407, 423
  - ; (Ende einer Anweisung) 106
  - ? (Konditionaloperator) 184
  - @ Fehler-Kontroll-Operator 189
  - \_\_autoload 404
  - \_\_autoload() 427
  - \_\_call 404
  - \_\_call() 427, 428
  - \_\_clone() 413
  - \_\_construct 404
  - \_\_construct() 406
  - \_\_destruct 404
  - \_\_destruct() 406
  - \_\_get() 427, 430
  - \_\_METHOD\_\_ 404, 441
  - \_\_set 404
  - \_\_set() 427, 430
  - \_\_toString() 427, 431
  - \_get 404
  - { } (Anweisungsblock) 109
  - || (Logisches Entweder Oder) 179
  - || (Logisches Oder) 178
  - + (Addition) 169
  - ++ (Inkrement) 186
  - < (Kleiner als) 173
  - <= (Kleiner gleich) 174
  - = (Zuweisung) 170
  - == (Gleichheit) 176
  - === (Strikte Gleichheit) 177
  - > (Größer als) 173
  - >= (Größer gleich) 174
- ## A
- abs() 279
  - abstract 403, 416
  - acos() 279
  - addslashes() 297
  - addslashes() 297, 298
  - ALTER
    - INDEX 602
    - TABLE 613
  - Änderungsprotokoll 687
  - Anweisungen 105
  - Anwendungsgebiete 34
  - Apache 39
  - Arithmetische
    - Divisionsoperator (/) 170
    - Moduloperator (%) 170
    - Multiplikationsoperator (\*) 170
    - Subtraktionsoperator (-) 169
  - Arithmetische
    - Additionsoperator (+) 169

- Array 131
  - array\_replace() 458, 460
  - array\_replace\_recursive() 458, 460
  - arsort() 251
  - asort() 251
  - Assoziativ 231
  - count() 246
  - current() 246
  - Datentyp 230
  - each() 248
  - Elemente 230
  - end() 249
  - erzeugen 230
  - extract() 254
  - Funktionen 238
  - in\_array() 278
  - indizierte 231
  - ksort() 252
  - leeren 233
  - linear 231
  - list() 247
  - löschen 233
  - max() 272
  - Mehrdimensional 233
  - min() 272
  - next() 249
  - pos() 246
  - prev() 249
  - reset() 249
  - rsort() 250
  - sizeof() 246
  - sonstige Funktionen 239
  - sort() 250
  - Sortierfunktionen 239, 250
  - Terminologie 230
  - uasort() 253
  - uksort() 253
  - usort() 253
- array\_change\_key\_case() 240
- array\_chunk() 240, 268
- array\_combine() 240, 267
- array\_count\_values() 240
- array\_diff() 241, 275
- array\_diff\_assoc() 240
- array\_diff\_key() 241
- array\_diff\_uassoc() 241
- array\_diff\_ukey() 241
- array\_fill() 241
- array\_filter() 241
- array\_flip() 241
- array\_intersect() 241, 274
- array\_intersect\_assoc() 241
- array\_intersect\_key() 241
- array\_intersect\_uassoc() 241
- array\_intersect\_ukey() 241
- array\_key\_exists() 242
- array\_keys() 242, 260
- array\_map() 242, 261
- array\_merge() 242, 264, 274
- array\_merge\_recursive() 242
- array\_multisort() 242, 269
- array\_pad() 242, 265
- array\_pop() 243, 259
- array\_push() 243, 258
- array\_rand() 243, 270
- array\_reduce() 243
- array\_reverse() 243, 260
- array\_search() 243, 271
- array\_shift() 243, 258
- array\_slice() 243, 265
- array\_splice() 243, 259
- array\_sum() 243
- array\_udiff() 244
- array\_udiff\_assoc() 244
- array\_udiff\_uassoc() 244
- array\_uintersect() 244
- array\_uintersect\_assoc() 244
- array\_uintersect\_uassoc() 244
- array\_unique() 244, 272, 274
- array\_unshift() 244, 257
- array\_values() 244, 266
- array\_walk() 238, 245
- array\_walk\_recursive() 245
- Arrays 229
  - arsort() 239, 251
  - asin() 279
  - asort() 239, 251
- Assoziatives Array 231
- atan() 279
- atan2() 279
- Attribute 571
- Ausdruck, Wahrheitsgehalt 104
- Ausdrücke 102
  - elementare 102
  - Funktionen 103
  - Konditional Operator 105
  - Post-Inkrement 103
  - Prä-Inkrement 103
  - Vergleichsausdrücke 104
  - Zusammengesetzt 102
  - Zuweisungen 104
- Ausnahmebehandlung 434
  - catch 436
  - debug\_backtrace() 438

geschachtelt 475  
 throw 436  
 try 436  
 AUTO\_INCREMENT 504, 611

**B**

BackUp 709  
 BACKUP, TABLE 714  
 base\_convert() 280  
 basename () 362  
 Bedingung 190  
 Bedingungen  
   if-Bedingung 190  
 Benutzer 700  
   ändern 700  
   anlegen 700  
   löschen 701  
 Bezeichner 148  
 Beziehung 568  
 bin2hex() 299  
 bindec() 280  
 BLOB 587  
 boolesche Werte 127  
 break 210  
 Browser 29

**C**

catch 404  
 ceil() 279  
 CGI 21  
   GET 21  
   POST 22  
 CHANGE 615  
 CHAR 587  
 chdir() 365  
 CHECK TABLE 689  
 checkdate() 287  
 chgrp () 362  
 chmod () 362  
 chop() 297  
 chown () 362  
 chr() 305  
 chunk\_split() 299, 300  
 class 404  
 class\_alias() 458  
 clearstatcache() 462  
 clone 403  
 close() 484  
 closedir () 368  
 Closures 454  
   callbacks 455  
   Objekte 457

Sichtbarkeitsbereich 454  
 use 454  
 Codezeile 106  
 Common Gateway Interface 21  
 compact() 245, 267  
 connection\_aborted() 358  
 connection\_status() 358  
 connection\_timeout() 358  
 const 403, 425  
 continue 210  
 convert\_cyr\_string() 299  
 convert\_uudecode() 299  
 convert\_uuencode() 299  
 Cookies  
   Array 348  
   löschen 349  
   mehrere Variablen 345  
   Namenskonflikte 345  
   setcookie() 343  
   Spezifikationen 341  
 copy() 462  
 cos() 279  
 count() 238, 246  
 count\_chars() 311, 313  
 CREATE  
   DATABASE 594  
   TABLE 597  
 create\_function() 452  
 crypt() 311, 313  
 current() 238, 246

**D**

date() 287, 289  
 date\_sunrise() 287  
 date\_sunset() 287  
 Datei, Download 388  
 Dateiattribut 372  
 Dateien 362  
   ändern 385  
   bearbeiten 385  
   Berechtigungen 366  
   CSV 382  
   Datensätze 376  
   entfernter Server 373  
   erzeugen 377  
   Funktionen 370  
   komprimiert 386  
   kopieren 379  
   lesen 372  
   löschen 379  
   Mustersuche 386  
   Operationen 370

- schreiben 372
- Serialisierung 380
- umbenennen 379
- Upload 393
- Verriegelung 381
- Zeilen auslesen 383
- Dateisystem 362
- Daten
  - \$QUERY\_STRING 338
  - Escape-Zeichen 338
  - Kodierung 337
  - URL 336
  - verschlüsseln 703
- Datenbank 28, 565
  - anlegen 594
  - anzeigen 595
  - Backup 537
  - Entwicklungsphasen 565
  - ERM 568
  - löschen 594
  - Primärschlüssel 572
  - Tabellen 583
  - Tests 719
- Datenbanken 31
  - Anwendungsgebiete 34
  - relationale 31
- Datenbankmanagementsysteme 566
- Datenbankmodellierung 89
- Datenbanktabelle
  - Felder ändern 521
  - Felder entfernen 522
  - Felder hinzufügen 520
- Datenbanktabellen
  - Beziehungen 568
  - Join 523, 525, 526
  - Outer Join 533
  - Self Join 531
  - Theta Join 529
  - WHERE 525
- Datenmodell 567
- Datensicherung 709
  - Provider 717
  - Strategien 709
- Datentyp 117
  - Aufzählung 590
  - Datums- und Zeitwerte 585
  - Mapping 590
  - Zahlen 585
  - Zeichenketten 584
- Datentypen 584
  - Array 131
  - Boolean 127
  - Grundtyp 117
  - NULL 136
  - Objekte 130
  - Referenztyp 117
  - Resource Typ 136
  - String 117
  - Zahlen 122
- DATETIME, DATE 589
- Datum
  - date() 289
  - Funktionen 287
  - getdate() 288
  - idate() 290
  - Ostern 286
  - setlocale() 296
- Datumfunktionen 285
  - Kalender 285
- Datums- und Zeitwerte 585
- DBMS 565
- DBMS-Systeme 565
- DCL-Befehle 580
- DDL 565
- DDL-Befehle 581
- debug\_backtrace() 438
- decbin() 280
- dechex() 280
- decoct() 280
- Decodierung 703
- Dedizierter Server 62, 557
- deg2rad() 280
- Dekrement 186
- Design Patterns 443
- dir () 368
- dirname() 362
- disk\_free\_space() 362
- disk\_total\_space() 363
- DML 565
- DML-Befehle 580
- DQL-Befehle 581
- DROP
  - DATABASE 595
  - TABLE 618
- E**
  - each() 238, 248
  - echo() 305
  - Editor 29
  - Eigener Webserver 62, 557
  - Eigenschaften, statisch 421
  - empty() 152
  - EMS SQL Manager 74
  - end() 238, 249

Entität 568  
 Entitätstyp 568  
 Entity-Relationship-Modell 568  
 Entwicklungsumgebungen 92  
 Entwurfsmuster 443  
   Nutzen 443  
   Regeln 443  
   Singleton 444  
 ENUM 590  
 ERM 568  
 Escape 118  
 Exception 434  
   geschachtelt 475  
 exp() 279  
 explode() 300  
 extends 415  
 externe Skripts 100  
 extract 512  
 extract() 239, 254

## F

false 127  
 fclose() 371  
 Fehlerbehandlung 434  
 Fehlermeldungen 501  
 Fehlerprotokoll 685  
 Felder 229  
 Feldtypen 584  
 feof() 378  
 Festkommazahl 588  
 fetch\_object() 484  
 fgetc() 371  
 fgetcsv() 371  
 fgetcsv() 462  
 fgets() 371  
 fgets() 374  
 fgetss() 371  
 file() 371  
 file() 371  
 file\_exists() 363  
 file\_get\_contents() 363  
 file\_get\_contents() 389  
 file\_put\_contents() 363  
 file\_put\_contents() 389  
 fileatime() 363  
 filegroup() 363  
 filemtime() 363  
 fileowner() 363  
 fileperms() 363  
 filesize() 363  
 filetype(); 363  
 final 403, 416  
 Fließkommazahlen 588  
 Fließkommazahlen Runden 281  
 flock() 371  
 floor() 279  
 FLUSH PRIVILEGES 699  
 flush() 305  
 fopen() 371  
 FOREIGN KEY 612  
 Foreign Keys 572  
 Formulare 317  
   Auswerten 322  
   Checkboxen 331  
   Dynamisch 326  
   Kontrollkästchen 331  
   mehrere Seiten 329  
   Optionsschalter 331  
   Radiobuttons 331  
   register\_globals 334  
   submit 332  
   Validation 333  
 Formularelemente 320, 325, 331  
 forward\_static\_call() 458  
 forward\_static\_call\_array() 459  
 fprintf() 305  
 fputs() 371  
 fread() 371  
 Fremdschlüssel 572, 612  
 fseek() 378  
 ftell() 378  
 ftell() 379  
 Functors 475  
 Funktionen  
   anonym 452  
   Argumente 212  
   closures 454  
   Dynamisch 217  
   Fehlercode 216  
   Funktionsobjekte 475  
   Lambda 452  
   lokale und globale Variablen 213  
   Parameter 212  
   Rekursiv 221  
   Rückgabewerte 216  
   verschachtelt 218  
 Funktionsobjekte 475  
 fwrite() 371

## G

Ganzzahlen 588  
 Garbage Collection 477  
 gc\_collect\_cycles() 459  
 gc\_disable() 459

- gc\_enable() 459
- gc\_enabled() 459
- GET 318
- get\_called\_class() 459
- get\_defined\_vars() 153
- get\_meta\_tags() 307
- getcwd () 365
- getdate() 287, 288
- gethostname() 459, 461
- getrandmax() 282
- gettimeofday() 292
- gettype() 143
- Gleichheitsoperatoren
  - Gleichheit (==) 176
  - Strikte Gleichheit (===) 177
  - Strikte Ungleichheit (!==) 177
  - Ungleichheit (!=) 176
- glob() 390
- gmdate() 287
- gmmktime() 292
- gmstrftime() 292
- GRANT 697
- Groß- und Kleinschreibung 109
- Gültigkeitsoperator 423
  
- H**
- Hashes 231
- header\_remove() 459, 462
- Heredoc 113, 120, 451
- hexdec() 280
- htmlentities() 307, 309
- HTML-Formulare 319
- htmlspecialchars() 308
- htmlspecialchars\_decode() 307
  
- I**
- idate() 287, 290
- IDE 92
- if-Anweisung 190
- ignore\_user\_abort() 359
- implements 403, 419
- implode() 300, 301
- in\_array() 245, 266
- include() 100
- include\_once() 101
- ini\_get\_all() 463
- Inkrement 186
- Installation
  - aktualisieren 553
  - MySQL 64
  - Unix/LINUX 544
  - Vorbereitung 37
  - Windows 547
- Installation-Kit 42
  - Apache2Triad 49
  - MAMP 52
  - Sicherheit 52
  - WAMP5 51
- instanceof 403, 441
- interface 403
- Interface 419
- Internet Service Provider 61
- Internet-Service-Provider 557
- Interzeptormethoden 427
- intl 479
- Is\_a() 465
- is\_array() 143
- is\_bool() 143
- is\_dir () 363
- is\_double() 143
- is\_executable () 363
- is\_file () 363
- is\_float() 143
- is\_int() 143
- is\_integer() 143
- is\_link () 363
- is\_long() 143
- is\_null() 143
- is\_numeric() 143
- is\_object() 143
- is\_readable () 363
- is\_real() 143
- is\_resource() 143
- is\_scalar() 143
- is\_string() 143
- is\_uploaded\_file () 363
- is\_writable () 363
- ISP 61, 557
- isset() 151
  
- J**
- join() 300
  
- K**
- key() 238, 247
- Klammern
  - Geschweifte Klammern 109
  - Runde Klammern 110
- Klassen, abstract 416
- Kodierung 337
- Kommentar 115
  - Einzeilig 116
  - Mehrzeilig 116
- Konditionaloperator (?) 184

- Konstanten 165
  - vordefiniert 165
- Kontrollstrukturen
  - break 210
  - continue 210
  - do-while-Schleife 202
  - foreach-Schleife 206
  - for-Schleife 203
  - if-Anweisung 190
  - if-else-Anweisung 193
  - if-else-if-Anweisung 195
  - switch-Anweisung 197
  - Verschachtelt 208
  - while-Schleife 200
- krsort() 239
- Krypto-Filesystem 704
- ksort() 239, 252
- KSql 83
- L**
- Lambda-Funktionen 452
- LAMP 41, 555
- Late Static Bindings 471
- lcfirst() 459, 462
- Leerzeichen 108
- levenshtein() 303
- libmysql 476
- list() 239, 247
- localtime () 291
- localtime() 294
- LOCK TABLES 710
- log() 279
- log10() 279
- Logfiles 360
- Log-Files 685
- Logische Operatoren
  - Logisches Entweder ODER (xor) 179
  - Logisches NICHT (!) 179
  - Logisches ODER (||) 178
  - Logisches UND (&&) 178
- Lokalisierung 551
- ltrim() 298
- M**
- Magische Methoden
  - \_\_autoload() 427
  - \_\_call() 427, 428
  - \_\_get() 427, 430
  - \_\_set() 427, 430
  - \_\_toString() 427, 431
- Mathematische
  - Konstanten 282
  - Umwandlungsfunktionen 279
  - Zufallszahlen 282
- Mathematisch, Runden 281
- Mathematische Funktionen 279
- max() 272, 279
- md5() 312
- md5\_file() 312
- Mehrdimensionales Array 233
- metaphone() 304
- Methoden
  - abstract 416
  - statisch 421
- microtime() 292
- min() 272, 279
- mkdir () 365
- mktime() 291, 293
- MODIFY 615
- move\_uploaded\_file() 397
- mt\_getrandmax() 283
- mt\_rand() 283
- mt\_srand() 282
- mysamchk 73, 562
- mysqli\_fetch\_object 514
- MySQL 481
  - Anwendungsgebiete 34
  - AUTO\_INCREMENT 504
  - Backup 537
  - Benutzer anlegen 496
  - Benutzer löschen 498
  - Benutzerverwaltung 495
  - Clients 74
  - Datenbank erstellen 502
  - Datenbanken auflisten 499
  - Datenbanken löschen 519
  - Datenbankverbindung 483
  - Datensätze ausgeben 511
  - Datensätze bearbeiten 509
  - Datensätze einfügen 507
  - Datensätze löschen 510
  - Datentypen 504, 584
  - Fehlermeldungen 501
  - Funktionsumfang 557
  - Installation 64, 482, 543
  - Installation prüfen 549
  - Installationspakete 548
  - JOIN 525
  - Kommandozeilenwerkzeuge 71
  - Konfigurationsdatei 550
  - Mehrere Server 553
  - Nutzerangaben ändern 498
  - Pfadangaben 551
  - Primärschlüssel 504

- Provider 557
- Relationen 525
- Releasenummer 65
- Schnittstellen 35
- Server 66, 544
- Spaltentypen 504
- Tabelle erstellen 503
- Tabellen ändern 520
- Tabellen entfernen 519
- Tabellenabfragen verknüpfen 523
- Tabellenanzahl 506
- Tabellenstruktur 522
- Unix/LINUX 66
- Verbindung 498
- Versionen 543
- Vorteile 32
- Windows 69
- Zugriffe 502
- MySQL Administrator 84
- MySQL Maestro 80
- MySQL Migration Toolkit 86
- MySQL Query Browser 88
- MySQL Turbo Manager 82
- MySQL Workbench 87
- mysql\_affected\_rows 491
- mysql\_change\_user 491
- mysql\_client\_encoding 491
- mysql\_close 491
- mysql\_connect 491
- mysql\_create\_db 491
- mysql\_data\_seek 491
- mysql\_db\_query 491
- mysql\_drop\_db 491
- mysql\_errno 491
- mysql\_error 491
- mysql\_escape\_string 491
- mysql\_fetch\_array 491
- mysql\_fetch\_field 492
- mysql\_fetch\_lengths 492
- mysql\_fetch\_object 492
- mysql\_fetch\_row 492
- mysql\_field\_flags 492
- mysql\_field\_len 492
- mysql\_field\_name 492
- mysql\_field\_seek 492
- mysql\_field\_table 492
- mysql\_field\_type 492
- mysql\_free\_result 492
- mysql\_get\_client\_info 492
- mysql\_get\_host\_info 493
- mysql\_get\_proto\_info 493
- mysql\_get\_server\_info 493
- mysql\_info 493
- mysql\_insert\_id 493
- mysql\_list\_dbs 493
- mysql\_list\_fields 493
- mysql\_list\_tables 493
- mysql\_num\_fields 493
- mysql\_num\_rows 494
- mysql\_pconnect 494
- mysql\_ping 494
- mysql\_query 494
- mysql\_real\_escape\_string 494
- mysql\_result 494
- mysql\_select\_db 494
- mysql\_stat 494
- mysql\_tablename 494
- mysql\_thread\_id 494
- mysql\_unbuffered\_query 495
- mysqldadmin 72, 562, 596
- mysqld\_multi 554
- mysqldump 73, 563
- mysqlhotcopy 74, 563
- MySQLi
  - affected\_rows 487, 489
  - autocommit 488
  - bind\_param 489
  - bind\_result 489
  - change\_user 488
  - character\_set\_name 488
  - close 488, 489, 490
  - close() 484
  - commit 488
  - connect 488
  - current\_field 490
  - data\_seek 489, 490
  - Datenbank erzeugen 485
  - Datenbanktabelle erzeugen 485
  - Datensätze auslesen 486
  - Datensätze hinzufügen 486
  - errno 487, 489
  - error 487, 489
  - execute 489
  - fetch 489
  - fetch\_assoc 490
  - fetch\_field 490
  - fetch\_field\_direct 490
  - fetch\_fields 490
  - fetch\_lengths 490
  - fetch\_object 490
  - fetch\_row 490
  - field\_count 487, 490
  - field\_seek 490
  - get\_client\_info 488

get\_client\_version 488  
 get\_host\_info 488  
 get\_metadata 489  
 host\_info 487  
 info 487, 488  
 init 488  
 insert\_id 487  
 Installation 482  
 kill 488  
 length 490  
 more\_results 488  
 multi\_query 488  
 next\_result 488  
 num\_rows 490  
 options 488  
 param\_count 489  
 ping 488  
 prepare 488, 489  
 protocol\_version 488  
 query 488  
 query() 484, 502  
 real\_connect 488  
 real\_query 489  
 Referenz 487  
 rollback 489  
 select\_db 489  
 send\_long\_data 489  
 send\_query 489  
 SQL-Abfragen 484  
 sqlstate 488, 489  
 ssl\_set 489  
 stat 489  
 stmt\_init 489  
 store\_result 489  
 thread\_id 488  
 thread\_safe 488, 489  
 use\_result 489  
 warning\_count 488  
 mysqli() 484  
 mysqli\_fetch\_all() 459  
 mysqli\_get\_connection\_stats() 459  
 mysqliimport 73, 563  
 mysqlnd 476  
 mysqlshow 73, 562  
 mysqltest 73, 563

## N

Namensräume 465  
 Namespaces 465  
 natcasesort() 239  
 natsort() 239  
 Navicat 77

new 405  
 next() 238, 249  
 nl2br() 308, 310, 463  
 Normalformen 575  
 Normalisierung 574  
 Nowdoc 452  
 NULL 136  
 number\_format() 312, 314

## O

Object 130  
 Objekt 130
 

- Eigenschaften 130
- Methoden 130
- Referenzen 426

 Objekte
 

- \_\_clone() 413
- klonen 411
- kopieren 411
- zählen 422

 octdec() 280  
 OOA 443  
 OOD 443  
 OOP
 

- \_\_METHOD\_\_ 441
- abstract 416
- clone 411
- Closures 457
- const 425
- copy by value 426
- Datenkapselung 407
- Dereferenzierung 439
- Entwurfsmuster 443
- extends 415
- final 416
- Gültigkeitsoperator 423
- instanceof 441
- Interface 419
- Klassenkonstanten 425
- Klassenvererbung 415
- late static bindings 471
- magische Methoden 427
- Namespaces 465
- object by reference 426
- parent 424
- private 408
- protected 408
- public 408
- Referenzen 426
- Rundgang 408
- Schnittstellen 419
- self 424

- Singleton 444
- späte Bindung 471
- static 421
- statische Eigenschaften 421
- statische Methoden 421
- Verweisoperator 423
- Zugriffsbeschränkung 407
- opendir () 368
- Operatoren 123, 166
  - @ 189
  - Arithmetische 169
  - Array 188
  - Assoziativität 168
  - Bitweise Operatoren 180
  - Gleichheitsoperatoren 176
  - Gruppierungsoperator () 185
  - Konditionaloperator (?) 184
  - Logische Operatoren 178
  - Objekterzeugungs-Operator (new) 187
  - Operatorliste 167
  - Vergleichsoperatoren 173
  - Vorrang 168
  - Zuweisungsoperator (=) 170
- OPTIMIZE TABLE 617, 692
- ord() 299
- Overloading
  - Klassenmethoden 428
  - Klassenmitglieder 430
- P**
- parent 407, 424
- parse\_ini\_file() 463
- parse\_ini\_string() 459
- parse\_str() 308
- parse\_url() 308, 310
- Passwortsystem 694
- perror 74, 563
- PHP
  - Anweisungen 105
  - Ausdrücke 102
  - bcmath-Funktionen 59
  - Browser-Einstellungen 59
  - Codezeile 106
  - Dateiendungen 24
  - Datenbanken 481
  - Datenbehandlung 55
  - Datentypen 117
  - Debugger-Optionen 58
  - echo 112
  - Erweiterungen 57
  - Extensions 57
  - Fehlerbehandlung 54
  - Funktionen 211
  - Heredoc 113, 120
  - HTML 25
  - include 100
  - Init-Datei 53
  - Installation 37, 40
  - Installation-Kits 42
  - Kommentar 115
  - Konfiguration 53, 62
  - Konfigurationsabbildungen 63
  - Konstanten 165
  - Kontrollstrukturen 190
  - Leistungsbegrenzungen 54
  - Linux-Installation 41
  - Logging-Funktion 58
  - Mail-Funktion 57
  - Migration 446
  - Moduleinstellungen 57
  - MySQL 481
  - MySQLi-Referenz 487
  - MySQL-Referenz 490
  - MySQL-Unterstützung 58
  - MySQL-Verbindung 498
  - Nowdoc 452
  - ODBC-Unterstützung 58
  - Operatoren 166
  - Operator-Rangfolge 167
  - Pfade 56
  - php.ini 53
  - phpinfo() 62
  - print 112
  - Provider 61
  - Referenzen 226
  - register\_globals 161
  - require 101
  - Session-Verwaltung 59
  - SGML-Stil 25
  - Sicherheit 59
  - Sprachoptionen 53
  - SQL-Optionen 58
  - String-Operator 184
  - Syntax 99
  - Systemprotokoll-Variablen 57
  - Überlegungen zur Installation 37
  - Variablen 145
  - Verzeichnisse 56
  - Webanwendung 26
  - Windows-Installation 38
  - Zuweisungen 111
- PHP 4, Anpassung 446
- PHP 5
  - Anpassung 446

- Destruktoren 406
- Klassen 404
- Konstruktoren 406
- Objekte erzeugen 405
- OOP Überblick 408
- parent 407
- Schlüsselwörter 446
- PHP 5.3, Neuerungen 451
- PHP als Apache-Modul 24, 41
- PHP als CGI-Programm 24, 41
- phpMyAdmin 75
- pos() 238, 246
- POST 317
- Postfixnotation 186
- pow() 279
- Präfixnotation 186
- Präzision 127
- prev() 238, 249
- Primärschlüssel 572
  - Regeln 572
- Primary Key 599
- print() 305
- printf() 305
- private 403, 408
- protected 403, 408
- Protokollierung 685
- Provider 61
  - Angebote 61
  - Dedizierter Server 62, 557
  - Eigener Webserver 62, 557
  - Webpace 62, 557
  - Zugangsdaten 61
- public 404, 408

## Q

- query() 484
- QUERY\_STRING 338
- Query-Protokoll 685
- quoted\_printable\_decode() 299
- quoted\_printable\_encode() 459
- quotemeta() 297

## R

- rad2deg() 280
- rand() 282
- range() 240, 256
- rawurldecode() 308, 310
- rawurlencode() 308, 309
- readdir () 368
- readfile () 371
- readfile() 371
- Referenzen 226

- aufheben 228
- pass-by-reference 228
- return-by-reference 228
- register\_globals 161, 334
- Sicherheit 162
- register\_shutdown\_function() 359
- Rekursion 221, 392
  - Fakultät 221
  - Türme von Hanoi 224
- Rekursive Funktionen 221
- Relation 568, 572
- Relationale Datenbanken 31
- Relationales Datenmodell 571
- RENAME 617
- REPAIR TABLE 691
- Replikationssystem 715
  - Benutzer 716
  - Installation 716
- require() 101
- require\_once() 101
- Reservierte Wörtern 110
- reset() 238, 249
- Resource Typ 136
- REVOKE 700
- rewind() 378
- rewinddir () 368
- rmdir () 365
- round() 280, 281, 463
- rsort() 239, 250
- rtrim() 298
- Runden
  - Genauigkeit 281

## S

- scandir () 368
- scandir() 390
- Schlüssel 569
- Schlüsselfelder 599
- Schlüsselwörter 110
- Schnittstellen 35
  - implements 419
  - interface 419
- SELECT INTO OUTFILE 715
- self 404, 424
- Semikola 106
- Servervariablen 359
  - Logfiles 360
- Session 349
  - Array 355
  - Cookies 356
  - Funktionen 352
  - GET/POST 357

- header() 358
- Konfiguration 350
- session\_destroy() 354
- session\_name() 356
- session\_start() 353
- uniqid() 351
- session\_cache\_expire() 353
- session\_cache\_limiter() 353
- session\_decode() 353
- session\_destroy() 353
- session\_encode() 353
- session\_get\_cookie\_params() 353
- session\_id() 353
- session\_is\_registered() 353
- session\_module\_name() 353
- session\_name() 353
- session\_regenerate\_id() 353
- session\_register() 350, 353
- session\_save\_path() 353
- session\_set\_cookie\_params() 353
- session\_set\_save\_handler() 353
- session\_start() 353
- session\_unregister() 350, 353
- session\_unset() 353
- session\_write\_close() 353
- Session-Management 349
- SET 590
- setcookie() 343
- setlocale() 296
- settype() 141
- sha1() 312
- sha1\_file() 312
- Shell 71, 560
- SHOW
  - COLUMNS 614
  - DATABASES 595
  - INDEX 602
  - TABLES 598
- SHOW STATUS 683
- shuffle() 240, 257
- Sicherheit 59, 683
  - Angriffsszenarien 60
  - Cross-Site Request Forgery 723
  - Cross-Site-Scripting 723
  - Gefahren 723
  - Gegenmaßnahmen 725
  - HTTP Response Splitting 724
  - Information Disclosure 723
  - Konzept 726
  - Parameterattacke 60
  - Pfadattacke 60
  - Probleme 59
  - Remote Command Execution 724
  - Schwachstellen 723
  - SQL-Injection 724
  - Webserver-Umgebung 61
- similar\_text() 303
- sin() 279
- sizeof() 238, 246
- sort() 239, 250
- soundex() 303
- Spaltentypen 504
- späte Bindung 471
- SPL 477
- Sprachanpassung 552
- sprintf() 305
- Spruchgenerator 384
- SQL, Standards 32
- Sqlite3 476
- SQL-Shell 71, 560
- SQLyog 79
- sqrt() 279
- rand() 282
- sscanf() 312
- Standard PHP Library 477
- static 404
- str\_getcsv() 459
- str\_ireplace() 298
- str\_pad() 311
- str\_repeat() 312
- str\_replace() 298, 299
- str\_shuffle() 312, 314
- str\_split() 312
- str\_word\_count() 312
- strcasecmp() 302
- strchr() 303
- strcmp() 302, 304
- strcspn() 303
- stream\_context\_create() 463
- stream\_context\_get\_params() 459
- stream\_context\_set\_default() 459
- stream\_supports\_lock() 459
- strftime() 292, 295
- String 117, 297
  - addslashes() 298
  - Ausgabe 305
  - chunk\_split() 300
  - count\_chars() 313
  - crypt() 313
  - echo() 306
  - Ersetzen 297
  - Escape 118
  - explode() 300
  - htmlentities() 309

htmlspecialchars() 308  
 implode() 301  
 nl2br() 310  
 number\_format() 314  
 parse\_url() 310  
 print() 306  
 printf() 305  
 rawurldecode() 310  
 rawurlencode() 309  
 sprintf() 305  
 str\_shuffle() 314  
 strcmp() 304  
 stripslashes() 298  
 strok() 301  
 Suchen 302  
 Teilen 299  
 Umwandeln 299  
 URL 307  
 Verbinden 299  
 Vergleichen 302  
 Stringfunktionen 297  
 strip\_tags() 308  
 stripslashes() 297  
 stripos() 302  
 stripslashes() 297, 298  
 stristr() 303, 463  
 strlen() 312  
 strnatcasecmp() 302  
 strnatcmp() 302  
 strpos() 302  
 strrchr() 303  
 strrev() 299  
 stripslashes() 302  
 strrpos() 302  
 strspn() 303  
 strstr() 303, 463  
 strtok() 300, 301  
 strtolower() 299  
 strtoupper() 299  
 strtr() 298  
 substr() 303  
 substr\_replace() 298  
 substr\_count() 303

## T

Tabelle

ändern 613  
 anlegen 597  
 kopieren 611  
 löschen 617  
 umbenennen 617  
 Tabellen 583

Indizes 600  
 vereinigen 533  
 verknüpft 523  
 Tabellentypen 602  
 Berkeley DB 608  
 Gemini 611  
 InnoDB 604  
 ISAM 604  
 MEMORY 607  
 MERGE 605  
 MyISAM 604  
 Transaktionsfähige 603  
 Vergleich 611  
 TABLE  
 ADD 614  
 ADD INDEX 615  
 ADD PRIMARY KEY 615  
 ADD UNIQUE 615  
 CHANGE 615  
 DROP 616  
 MODIFY 615  
 RENAME 614  
 Tabulatoren 108  
 tan() 279  
 TEXT 587  
 throw 404  
 TIME 589  
 time() 292  
 TIMESTAMP 589  
 timezone\_version\_get() 459  
 touch() 364  
 trim() 298  
 true 127  
 try 404  
 Tupel 571  
 Typkonvertierung 139  
 automatisch 137

## U

uasort() 239, 253  
 Überwachung 683  
 ucfirst() 299  
 ucwords() 299  
 uksort() 239, 253  
 umask() 364, 367  
 UML 443  
 Umwandlungsfunktionen 279  
 UNION 533  
 UNIQUE 615  
 UNLOCK TABLES 711  
 unset() 152  
 Upload 717

urldecode() 308  
 urlencode() 308  
 usort() 239, 253

**V**

var\_dump() 142  
 VARCHAR 587  
 Variablen 145  
   Aufbau 149  
   Bezeichner 148  
   Definition 146  
   Dynamisch 157  
   global 156  
   Gültigkeitsbereich 155  
   L-/R-Wert 147  
   löschen 152  
   prüfen 152  
   register\_globals 161  
   Speicher 146  
   Speicherklassen 156  
   Typkonvertierung 137, 139  
   vordefiniert 159  
   Werte 148  
 Variablenfunktionen 220  
 Variablen, PHP 160  
 Verbindungsstatus 358  
 Vergleichsoperatoren  
   Größer als (>) 173  
   Größer oder gleich (>=) 174  
   Kleiner als (<) 173  
   Kleiner oder gleich (<=) 174  
 Verschlüsselung 703  
 Verwaltung 683  
 Verweisoperator 423  
 Verzeichnisoperationen 365  
 Verzeichnisse 362  
   auslesen 368  
   Berechtigungen 366  
   glob() 390  
 Verzweigungen  
   if-Verzweigung 190  
 vprintf() 305  
 vsprintf() 305

**W**

WAMP 38, 556  
 Webanwendung 26

Websserver 28  
 Webspaces 62, 557  
 Wertebehälter 145  
 wordwrap() 308

**X**

XAMPP 42  
   Deinstallation 45  
   Installation 45  
   starten 47  
   stoppen 47  
   Testbetrieb 49

**Y**

YEAR 590

**Z**

Zahlen 122, 585  
   Fließkommazahlen 122, 123  
   Float 122  
   Hexadezimalzahl 123  
   Integer 122  
 Zeichenketten 584  
   Ausgabe 305  
   Ersetzen 297  
   Suchen 302  
   Teilen 299  
   Umwandeln 299  
   URL 307  
   Verbinden 299  
   Vergleichen 302  
 Zeichensatz 551  
 Zeilentrenner 108  
 Zeit  
   localtime() 294  
   microtime() 292  
   mktime() 293  
   setlocale() 296  
   strftime() 295  
 Zeitfunktionen 285, 291  
 Zend Studio 93  
 Zufallseinträge ohne Wiederholung 284  
 Zufallszahlen 282  
 Zufallszahlen ohne Wiederholung 284  
 Zuweisungen 170  
   mit Operation 171

# PHP 5 MySQL 5

*PHP und MySQL – das ist das bewährte Erfolgsduo für dynamische Internetanwendungen aller Art. Erfolgsautor Matthias Kannengiesser vermittelt in dieser aktualisierten Version seines Standardwerks das Profiwissen, das Sie für die anspruchsvolle Webprogrammierung brauchen. Ob objektorientierte Programmierung mit PHP, das Arbeiten mit Datenbanktabellen und Verzeichnissen oder komplexe Anwendungen wie Webshops oder Foren – Matthias Kannengiesser kennt die Lösung.*

## ▶ Alle PHP-Neuheiten

PHP 5.3 ist die umfangreichste Aktualisierung der Sprache seit der Einführung von PHP 5. Die neu eingeführten Befehle unterstützen vor allem die Entwicklung komplexer Webanwendungen. Eine besonders wichtige Neuerung sind die Namensräume (Namespaces), die dabei helfen, auch umfangreiche Programme einfach und klar zu strukturieren. Die neu eingeführten Lambda-Funktionen und Closures unterstützen darüber hinaus die Entwicklung von sicherem und zuverlässigem Code und erleichtern das Finden von Fehlern erheblich. Matthias Kannengiesser zeigt, wie Sie diese neu eingeführten Features sinnvoll und produktiv einsetzen.

## ▶ Echtes PHP-Profiwissen

Der Grund für den Erfolg der Skriptsprache PHP ist ihre Einfachheit. Der Autor demonstriert in diesem Buch, wie Sie eine PHP-Umgebung auf Ihrem PC einrichten und mit einfachen Praxisbeispielen schnelle Erfolgserlebnisse erzielen. Darauf aufbauend, vermittelt er das Profiwissen, das Entwickler für komplexe Webanwendungen brauchen. So nehmen zum Beispiel die objektorientierten Features von PHP sowie die Fehlerbehandlung und -vermeidung breiten Raum ein.

## ▶ Mit Datenbanken umgehen

Die Grundlage jeder dynamischen Webanwendung ist eine Datenbank, MySQL ist hier der Standard. Dieses Buch zeigt, wie Sie mit MySQL Datenbanklösungen programmieren, die zu Ihren Anforderungen passen. Sie erfahren, wie Sie Datenmodelle und Datenbanktabellen erstellen, Daten einfügen, auslesen, aktualisieren und wieder löschen können. Matthias Kannengiesser zeigt außerdem, wie Sie Ihre Datenbank von PHP aus ansprechen und die gespeicherten Informationen auf Ihrer Website jederzeit korrekt darstellen.

## Aus dem Inhalt:

- Funktionsweise von Webanwendungen
- Installation von PHP und MySQL
- PHP-Programmierwerkzeuge und Datenbanktools im Überblick
- Grundlagen von PHP und MySQL
- Cookies, Session-IDs und Logfiles erstellen
- Objektorientierte Programmierung mit PHP
- PHP 5.3: die neu eingeführten Features
- Datenbanken und Tabellen erstellen
- Datenbanktabellen verknüpfen, vereinigen und löschen
- Daten einfügen, auslesen und löschen
- Informationen aus anderen Programmen übernehmen
- Datenbanken mit PHP ansprechen
- Backup und Datensicherung
- Bonuskapitel auf der Buch-CD: Internet-Gästebücher und Besucherzähler selbst programmieren

## Über den Autor:

Matthias Kannengiesser ist Diplominformatiker und arbeitet als IT-Projektmanager, -Consultant und -Fachdozent für namhafte Unternehmen im In- und Ausland. Seit mehr als zehn Jahren hält er Seminare, Workshops und Vorträge zu den Themen PHP und Datenbankprogrammierung, ActionScript, Flash, Lingo und JavaScript und schreibt erfolgreiche Bücher und Magazinbeiträge zu diesen Themen. Er lebt in San Jose im US-Bundesstaat Kalifornien.

## Auf CD-ROM:

- PHP/MySQL-Umgebung XAMPP 1.7.1 für Windows, Linux und Mac OS X
- PHP-Entwicklungsplattform Aptana Studio 2.0 für Windows, Linux und Mac OS X
- MySQL Workbench für Windows, Linux und Mac OS X
- Zahlreiche Bonuskapitel, unter anderem eine umfangreiche MySQL-Referenz
- Sämtliche Programmierbeispiele



9 783645 600101

30,- EUR [D]

ISBN 978-3-645-60010-1

Besuchen Sie unsere Website  
[www.franzis.de](http://www.franzis.de)