

*Beispiele und Lösungen für
PHP-Programmierer*

3. Auflage
Aktuell zu PHP 5.3



PHP 5

Kochbuch

David Sklar & Adam Trachtenberg

*Überarbeitung und Aktualisierung von Carsten Lucke,
Matthias Brusdeylins, Ulrich Speidel & Stephan Schmidt*

O'REILLY®

Einleitung	XVII
1 Strings	1
1.0 Einführung	1
1.1 Auf Teil-Strings zugreifen	4
1.2 Teile von Strings ersetzen	5
1.3 Einen String zeichenweise verarbeiten	7
1.4 Einen String wort- oder zeichenweise umkehren	8
1.5 Tabulatoren expandieren und komprimieren	9
1.6 Die Groß- und Kleinschreibung in Texten ändern	11
1.7 Funktionen und Ausdrücke in Strings interpolieren	13
1.8 Leerzeichen aus einem String entfernen	14
1.9 Kommaseparierte Daten zerlegen	15
1.10 Begrenzte Daten mit fester Länge zerlegen	17
1.11 Strings aufteilen	19
1.12 Text an bestimmten Zeilenlängen umbrechen	22
1.13 Binärdaten in einem String speichern	23
2 Zahlen	27
2.0 Einführung	27
2.1 Prüfen, ob ein String eine gültige Zahl enthält	28
2.2 Fließkommazahlen vergleichen	29
2.3 Fließkommazahlen runden	30
2.4 Mit Bereichen von Integer-Zahlen arbeiten	31
2.5 Zufallszahlen innerhalb eines Bereichs generieren	32
2.6 Verzerrte Zufallszahlen generieren	34
2.7 Logarithmen berechnen	35
2.8 Potenzen berechnen	36

2.9	Zahlen formatieren	37
2.10	Den Plural korrekt ausgeben	38
2.11	Trigonometrische Funktionen berechnen	39
2.12	Trigonometrische Funktionen mit Graden anstelle von Bogenmaßen berechnen	40
2.13	Mit sehr großen oder kleinen Zahlen arbeiten	41
2.14	Zwischen Zahlensystemen konvertieren	43
2.15	Mit anderen Zahlen als Dezimalzahlen rechnen	44
3	Datum und Zeit	47
3.0	Einführung	47
3.1	Das aktuelle Datum und die aktuelle Zeit feststellen	49
3.2	Datums- und Zeitbestandteile in einen Epochen-Zeitstempel konvertieren	52
3.3	Einen Epochen-Zeitstempel in Zeit- und Datumsbestandteile konvertieren	54
3.4	Datum oder Zeit in einem bestimmten Format ausgeben	55
3.5	Die Differenz zwischen zwei Datumswerten berechnen	59
3.6	Den Abstand zwischen zwei Datumswerten über julianische Tage ermitteln	61
3.7	Den Tag der Woche, des Monats, des Jahres oder die Kalenderwoche des Jahres ermitteln	64
3.8	Start- und Enddatum einer Woche errechnen	66
3.9	Ein Datum validieren	67
3.10	Datums- und Zeitwerte aus Strings lesen	69
3.11	Addition und Subtraktion mit einem Datum	71
3.12	Die Zeit mit Zeitzonen berechnen	73
3.13	Geografische Lageinformationen zu einer Zeitzone bestimmen	79
3.14	Die Sommerzeit berücksichtigen	80
3.15	Zeitangaben mit hoher Genauigkeit generieren	82
3.16	Periodisch wiederkehrende Ereignisse berechnen	83
3.17	Andere Kalender als den gregorianischen verwenden	86
3.18	Programm: Kalender	87
4	Arrays	91
4.0	Einführung	91
4.1	Ein Array anlegen, das nicht mit dem Element 0 beginnt	94
4.2	Mehrere Array-Elemente unter einem Schlüssel speichern	95
4.3	Ein Array mit einer Folge von Integer-Werten initialisieren	96
4.4	Ein Array durchlaufen	97
4.5	Elemente aus einem Array löschen	100

4.6	Die Größe eines Arrays ändern	102
4.7	Ein Array an ein anderes anfügen	104
4.8	Ein Array in einen String verwandeln	106
4.9	Ein Array mit Kommata ausgeben	108
4.10	Prüfen, ob sich ein Schlüssel in einem Array befindet	109
4.11	Prüfen, ob sich ein Element in einem Array befindet	110
4.12	Die Position eines Elements in einem Array feststellen	111
4.13	Elemente finden, die einer bestimmten Prüfung standhalten	113
4.14	Das Array-Element mit dem größten oder kleinsten Wert finden	114
4.15	Ein Array umkehren	115
4.16	Ein Array sortieren	116
4.17	Ein Array über ein berechnetes Feld sortieren	117
4.18	Mehrere Arrays sortieren	119
4.19	Ein Array mithilfe einer Methode statt einer Funktion sortieren	121
4.20	Ein Array in eine zufällige Reihenfolge bringen	122
4.21	Einen Kartenstapel mischen	123
4.22	Doppelte Elemente aus einem Array entfernen	124
4.23	Die Vereinigungs-, Schnitt- oder Differenzmenge zweier Arrays ermitteln	125
4.24	Alle Elementkombinationen eines Arrays finden	127
4.25	Alle Permutationen eines Arrays finden	129
4.26	Eine Funktion auf jedes Element eines Arrays anwenden	132
4.27	Echte Objekte als Schlüssel von Arrays verwenden	134
4.28	Ein Objekt wie ein Array auftreten lassen	137
4.29	Programm: Ein Array horizontal angeordnet in einer HTML-Tabelle ausgeben	140
5	Variablen	143
5.0	Einführung	143
5.1	Die Verwechslung von == und = vermeiden	144
5.2	Einen Vorgabewert festlegen	145
5.3	Werte ohne Hilfe von temporären Variablen austauschen	146
5.4	Einen dynamischen Variablennamen erzeugen	147
5.5	Statische Variablen verwenden	149
5.6	Variablen in mehreren Prozessen gemeinsam verwenden	150
5.7	Komplexe Daten als String kapseln	152
5.8	Variableninhalte als Strings ausgeben	154
6	Funktionen	159
6.0	Einführung	159
6.1	Auf Funktionsparameter zugreifen	160

6.2	Vorgabewerte für Funktionsparameter festlegen	161
6.3	Werte als Referenzen übergeben	163
6.4	Benannte Parameter verwenden	164
6.5	Funktionen mit einer variablen Anzahl von Argumenten verwenden	166
6.6	Werte per Referenz zurückgeben	169
6.7	Mehr als einen Wert zurückgeben	169
6.8	Bestimmte Rückgabewerte überspringen	171
6.9	Fehlermeldungen zurückgeben	172
6.10	Variable Funktionen aufrufen	174
6.11	Innerhalb einer Funktion auf eine globale Variable zugreifen	176
6.12	Dynamische Funktionen erzeugen	178
6.13	Objekt-Datentypen für Funktionsparameter vorschreiben	179
7	Klassen und Objekte	181
7.0	Einführung	181
7.1	Objekte instantiieren	188
7.2	Objektconstructoren definieren	189
7.3	Destruktoren definieren	190
7.4	Zugriffskontrolle implementieren	192
7.5	Änderungen an Klassen und Methoden verhindern	195
7.6	String-Darstellungen für Objekte definieren	196
7.7	Interfaces definieren	198
7.8	Eine abstrakte Basisklasse definieren	201
7.9	Mit Namespaces Kollisionen zwischen Klassennamen verhindern ...	203
7.10	Namespace-Aliase – weniger Tipparbeit bei Verwendung von Namensräumen	205
7.11	Funktionen und Klassen aus dem globalen Namensraum verwenden	207
7.12	Objektreferenzen zuweisen	208
7.13	Objekte klonen	209
7.14	Callback-Funktionen mit einem Zustandsgedächtnis programmieren	212
7.15	Den Zugriff auf Eigenschaften abfangen	214
7.16	Methoden auf Objekten aufrufen, die von einer anderen Methode geliefert werden	219
7.17	Zusammengesetzte Klassen verschmelzen	219
7.18	Auf überschriebene Methoden zugreifen	224
7.19	Methodenpolymorphie einsetzen	226
7.20	Klassenkonstanten definieren	228
7.21	Statische Eigenschaften und Methoden definieren	230

7.22	Die Objektserialisierung steuern	233
7.23	Objektintrospektion	235
7.24	Prüfen, ob ein Objekt eine Instanz einer bestimmten Klasse ist	239
7.25	Klassendateien bei der Instantiierung von Objekten automatisch laden	242
7.26	Mehrere Autoload-Handler definieren	244
7.27	Objekte dynamisch instantiieren	246
7.28	Eine Anwendung: whereis	248
8	Effizienter Umgang mit Daten	251
8.0	Einführung	251
8.1	Über die Eigenschaften eines Objekts iterieren	254
8.2	Einfache Objekt-Iteration mit IteratorAggregate und ArrayObject	255
8.3	Einen eigenen Iterator implementieren	256
8.4	Einen eigenen Wrapper für Streams schreiben	260
8.5	Einen Stream filtern	264
8.6	Eigene Filter schreiben	266
8.7	Performancegewinn mit Arrays fester Größe erzielen	268
8.8	Standard-Datenstrukturen nicht neu erfinden – Queues, Stacks und Co.	269
9	Fehlerbehandlung mit Exceptions	273
9.0	Einführung	273
9.1	Fehlermeldungen vor Anwendern verbergen	275
9.2	Einstellungen zur Fehlerbehandlung vornehmen	276
9.3	Eine benutzerdefinierte Funktion zur Fehlerbehandlung verwenden	279
9.4	Fehler protokollieren	280
9.5	Debug-Informationen protokollieren	282
9.6	PHP-Fehler- und Warnmeldungen in Ausnahmen umwandeln	284
9.7	Ausnahmen abfangen	285
9.8	Eigene Ausnahmen werfen	287
9.9	Klassenabhängiges Exception-Handling	289
9.10	Vordefinierte Exception-Klassen für alle Lebenslagen	290
9.11	Ungefangene Exceptions zentral behandeln	292
9.12	Einen Stacktrace ausgeben	294
10	Web-Grundlagen	297
10.0	Einführung	297
10.1	Cookies setzen	298

10.2	Cookie-Werte lesen	300
10.3	Cookies löschen	301
10.4	Zu einer anderen Adresse umleiten	302
10.5	Sitzungen verfolgen	303
10.6	Sessions in einer Datenbank speichern	304
10.7	Verschiedene Browser erkennen	309
10.8	Einen GET-Query-String bilden	310
10.9	HTTP-Basic- oder -Digest-Authentifizierung einsetzen	312
10.10	Cookie-Authentifizierung verwenden	317
10.11	Ausgaben vorzeitig an den Browser senden	319
10.12	Ausgaben an den Browser zwischenspeichern	320
10.13	Web-Ausgaben mit gzip komprimieren	322
10.14	Den Fehler »headers already sent« vermeiden	322
10.15	Umgebungsvariablen lesen	324
10.16	Umgebungsvariablen setzen	325
10.17	Konfigurationsvariablen lesen	326
10.18	Konfigurationsvariablen setzen	328
10.19	Innerhalb von Apache kommunizieren	328
10.20	Code-Profile generieren	330
10.21	Geänderte Dateien herunterladen und unveränderte vom Browser cachen lassen	333
10.22	Programm: (De-)Aktivator für Website-Konten	336
10.23	Programm: Störungsprüfer	338
11	Formulare	345
11.0	Einführung	345
11.1	Formulareingaben verarbeiten	347
11.2	Formulareingaben prüfen	349
11.3	Mit mehrseitigen Formularen arbeiten	352
11.4	Formulare mit erhaltenen Informationen und Fehlermeldungen erneut anzeigen	355
11.5	Mehrfaches Absenden desselben Formulars verhindern	358
11.6	Hochgeladene Dateien verarbeiten	360
11.7	Die Formularverarbeitung durch PHP absichern	363
11.8	Steuerzeichen in Benutzerdaten durch Escape-Sequenzen ersetzen	365
11.9	Mit Formularvariablen arbeiten, deren Name einen Punkt enthält	366
11.10	Formularelemente mit Mehrfachoptionen verwenden	367
11.11	Drop-down-Menüs auf Basis des aktuellen Datums erzeugen	368

12 Datenbankzugriff	371
12.0 Einführung	371
12.1 Textdateien als Datenbanken verwenden	374
12.2 DBM-Datenbanken verwenden	376
12.3 Eine SQLite-Datenbank verwenden	380
12.4 Mit einer SQL-Datenbank verbinden	382
12.5 Eine SQL-Datenbank abfragen	384
12.6 Zeilen ohne eine Schleife abrufen	387
12.7 Daten in einer SQL-Datenbank modifizieren	389
12.8 Abfragen effizient wiederholen	391
12.9 Ermitteln, wie viele Zeilen eine Abfrage geliefert hat	394
12.10 Anführungszeichen maskieren	395
12.11 Debugging-Informationen und Fehler protokollieren	397
12.12 Eindeutige Identifikationsnummern erstellen	399
12.13 Abfragen dynamisch aufbauen	401
12.14 Paginierte Links für eine Gruppe von Datensätzen anzeigen	406
12.15 Ergebnisse und Abfragen cachen	409
12.16 An beliebiger Stelle eines Programms auf eine Datenbank- verbindung zugreifen	412
12.17 Programm: Ein Thread-basiertes Forum	414
13 Web-Automatisierung	423
13.0 Einführung	423
13.1 Eine URL mit der GET-Methode abrufen	425
13.2 Eine URL mit der POST-Methode abrufen	427
13.3 Eine URL mit Cookies abrufen	428
13.4 Eine URL mit Headern abrufen	430
13.5 Eine URL über eine beliebige HTTP-Methode abrufen	432
13.6 URL-Anforderung mit Timeout	434
13.7 Eine HTTPS-URL abrufen	436
13.8 Den Datenaustausch auf HTTP-Ebene debuggen	437
13.9 Eine Webseite mit Markup versehen	440
13.10 Links aus einer HTML-Datei extrahieren	442
13.11 ASCII in HTML konvertieren	443
13.12 HTML in ASCII konvertieren	444
13.13 HTML- und PHP-Tags entfernen	446
13.14 Die Protokolldatei eines Webservers analysieren	446
13.15 Auf eine AJAX-Anfrage antworten	449
13.16 Zusammenarbeit mit JavaScript	451
13.17 Programm: Veraltete Links finden	455
13.18 Programm: Aktualisierte Links herausfinden	457

14 XML	461
14.0 Einführung	461
14.1 XML manuell generieren	464
14.2 XML mit DOM generieren	466
14.3 XML-Dokumente mit xmlWriter generieren	470
14.4 Komplexe XML-Dokumente einlesen (DOM)	473
14.5 Große XML-Dokumente einlesen (SAX)	476
14.6 XML mit SimpleXML parsen	481
14.7 Daten zwischen DOM und SimpleXML austauschen	485
14.8 Große XML-Dokumente einlesen (xmlReader)	488
14.9 XML mit XSLT transformieren	494
14.10 PHP-Funktionen in XSL-Stylesheets verwenden	497
14.11 Informationen aus einem XML-Dokument selektieren (XPath)	501
14.12 XML-Dokumente für Menschen lesbar machen	506
14.13 XML-Dokumente aus PHP-Datenstrukturen erzeugen	510
14.14 XML-Dokumente in PHP-Arrays oder Objekte einlesen	513
14.15 XML-Dokumente validieren	518
14.16 Die Inhaltskodierung steuern	520
14.17 XSLT-Parameter aus PHP setzen	522
15 Webservices	525
15.0 Einführung	525
15.1 XML-RPC-Anfragen senden	526
15.2 XML-RPC-Anfragen empfangen	529
15.3 XML-RPC-Anfragen mit der XMLRPCi-Erweiterung senden	532
15.4 SOAP-Anfragen mit einem WSDL-Dokument senden	534
15.5 SOAP-Anfragen ohne ein WSDL-Dokument senden	538
15.6 SOAP-Anfragen empfangen	541
15.7 RSS-Feeds lesen	545
15.8 REST-Anfragen senden	548
15.9 Daten mit WDDX austauschen	554
16 Reguläre Ausdrücke	557
16.0 Einführung	557
16.1 Von ereg zu preg wechseln	561
16.2 Wörter suchen	563
16.3 Den n-ten Match finden	564
16.4 Zwischen gierigem und nicht-gierigem Matching wählen	565
16.5 E-Mail-Adressen validieren	567
16.6 Alle zu einem Muster passenden Zeilen in einer Datei finden	570

16.7	Text innerhalb von HTML-Tags finden	571
16.8	In regulären Ausdrücken Sonderzeichen verwenden	573
16.9	Datensätze lesen, bei denen ein Muster als Separator dient	575
16.10	Verhindern, dass Klammern Text fangen	576
16.11	In einem regulären Ausdruck eine PHP-Funktion nutzen	578
17	Verschlüsselung und Sicherheit	583
17.0	Einführung	583
17.1	Passwörter aus den Dateien Ihrer Site heraushalten	585
17.2	Daten durch Kodierung verschleiern	586
17.3	Daten durch Prüfsummen verifizieren	587
17.4	Passwörter speichern	589
17.5	Überprüfung der Passwortsicherheit	590
17.6	Was tun bei verlorenen Passwörtern?	592
17.7	Daten ver- und entschlüsseln	594
17.8	Verschlüsselte Daten in einer Datei oder Datenbank speichern	599
17.9	Verschlüsselte Daten gemeinsam mit einer anderen Website nutzen	602
17.10	SSL ermitteln	604
17.11	E-Mail mit GPG verschlüsseln	605
17.12	Metazeichen der Shell mit Escape-Zeichen versehen	607
17.13	Session-Fixierung verhindern	608
17.14	Sich gegen Formular-Spoofing schützen	609
17.15	Sicherstellen, dass Eingaben gefiltert werden	611
17.16	Cross-Site-Scripting verhindern	611
17.17	SQL-Injection verhindern	612
18	Grafik	615
18.0	Einführung	615
18.1	Linien, Rechtecke und Vielecke zeichnen	619
18.2	Bogen, Ellipsen und Kreise zeichnen	620
18.3	Unterbrochene Linien zeichnen	622
18.4	Text zeichnen	624
18.5	Zentrierten Text zeichnen	627
18.6	Dynamische Bilder zusammensetzen	632
18.7	Eine transparente Farbe ermitteln und einstellen	634
18.8	Programm: Heraufgeladene Digitalfotos auf Webformat verkleinern	635
18.9	Grafiken geschützt ausgeben	638
18.10	Programm: Aus Umfrageergebnissen Balkendiagramme erstellen	640

19	Internationalisierung und Lokalisierung	645
19.0	Einführung	645
19.1	Vorhandene Locales auflisten	647
19.2	Ein bestimmtes Locale verwenden	648
19.3	Das Default-Locale setzen	649
19.4	Textmeldungen lokalisieren	650
19.5	Datum und Uhrzeiten lokalisieren	654
19.6	Lokalisierung von Währungen	656
19.7	Bilder lokalisieren	659
19.8	Eingebundene Dateien lokalisieren	660
19.9	Lokalisierungsressourcen verwalten	661
19.10	gettext verwenden	663
19.11	Unicode-Zeichen lesen und ausgeben	665
19.12	Die Zeichenkodierung ausgehender Daten setzen	666
19.13	Die Zeichenkodierung eingehender Daten setzen	667
20	Internetdienste	669
20.0	Einführung	669
20.1	E-Mails senden	670
20.2	MIME-Mail senden	673
20.3	E-Mail mit IMAP oder POP3 lesen	676
20.4	Nachrichten an Usenet-Newsgruppen senden	679
20.5	Usenet-Nachrichten lesen	681
20.6	Dateien mit FTP herauf- und herunterladen	686
20.7	Adressen über LDAP abfragen	689
20.8	LDAP zur Benutzer-Authentifizierung verwenden	691
20.9	DNS-Lookups ausführen	693
20.10	Überprüfen, ob ein Host erreichbar ist	695
20.11	Informationen über einen Domainnamen herausfinden	697
21	Dateien	699
21.0	Einführung	699
21.1	Eine lokale Datei erstellen oder öffnen	703
21.2	Eine temporäre Datei erstellen	704
21.3	Eine Datei auf einem entfernten Server öffnen	705
21.4	Eine Datei in einen String einlesen	707
21.5	Einen String in eine Datei schreiben	708
21.6	Die Zeilen, Absätze oder Datensätze in einer Datei zählen	709
21.7	Jedes Wort einer Datei verarbeiten	711
21.8	Eine bestimmte Zeile einer Datei einlesen	713

21.9	Eine Datei zeilen- oder absatzweise in rückwärtiger Reihenfolge bearbeiten	714
21.10	Eine Zeile per Zufall aus einer Datei auswählen	714
21.11	Alle Zeilen einer Datei in eine Zufallsreihenfolge bringen	715
21.12	Textfelder variabler Länge verarbeiten	716
21.13	Konfigurationsdateien einlesen	717
21.14	Von einer bestimmten Stelle einer Datei lesen oder an eine bestimmte Stelle einer Datei schreiben	720
21.15	Die letzte Zeile einer Datei entfernen	721
21.16	Eine Datei an ihrem Platz ohne eine temporäre Datei ändern	723
21.17	Gepufferte Ausgabedaten in eine Datei schreiben	725
21.18	An viele Datei-Handles gleichzeitig schreiben	725
21.19	Einem Programm Eingabedaten durchgeben	727
21.20	Die Standardausgabe eines Programms lesen	727
21.21	Den Standardfehlerkanal eines Programms einlesen	729
21.22	Eine Datei sperren	730
21.23	Komprimierte Dateien lesen und schreiben	733
21.24	Programm: Unzip	735
22	Verzeichnisse	737
22.0	Einführung	737
22.1	Zeitstempel auslesen und setzen	740
22.2	Auf Dateieinformationen zugreifen	741
22.3	Dateiberechtigungen oder Dateieigentümerschaft ändern	743
22.4	Einen Dateinamen in seine Bestandteile zerlegen	744
22.5	Eine Datei löschen	746
22.6	Eine Datei kopieren oder verschieben bzw. umbenennen	746
22.7	Alle Dateien in einem Verzeichnis verarbeiten	747
22.8	Alle Dateinamen finden, die einem Muster entsprechen	749
22.9	Alle Dateien in einem Verzeichnis rekursiv verarbeiten	750
22.10	Dateien eines Verzeichnisses filtern	751
22.11	Neue Verzeichnisse erstellen	753
22.12	Ein Verzeichnis und seinen Inhalt entfernen	753
22.13	Programm: Eine Auflistung des Webserver-Verzeichnisses erstellen	755
22.14	Programm: Site-Suche	760
23	PHP auf der Kommandozeile	765
23.0	Einführung	765
23.1	Programmparameter parsen	767

23.2	Programmparameter mit Console_Getopt oder Console_Getargs parsen	768
23.3	Von der Tastatur lesen	775
23.4	Passwörter einlesen	777
23.5	Die Ausgabe eines Kommandozeilen-Befehls weiterverarbeiten	779
23.6	Dateien zeilenweise verarbeiten	781
23.7	Prozesse forken	784
23.8	Einen Server programmieren	787
24	PEAR und PHAR	793
24.0	Einführung	793
24.1	PEAR installieren	795
24.2	Den PEAR Package Manager verwenden	798
24.3	PEAR-Pakete installieren und deinstallieren	801
24.4	PEAR-Pakete upgraden	803
24.5	PECL-Pakete installieren	804
24.6	Pakete aus anderen Channels installieren	806
24.7	PEAR in Shared-Hosting-Umgebungen installieren	810
24.8	Eigene PEAR-Pakete erstellen	816
24.9	Eigene Pakete über einen Channel-Server vertreiben	822
24.10	Ein PHP-Archiv (PHAR) erstellen	833
24.11	Auf Inhalte in einem PHP-Archiv (PHAR) zugreifen	835
24.12	Ein PHP-Archiv (PHAR) direkt ausführen	837
	Index	839

11.0 Einführung

Das Geniale an PHP ist die nahtlose Integration der Formularvariablen in die Programme. Dadurch wird Webprogrammierung elegant und einfach, und zwar vom Webformular über den PHP-Code bis zur HTML-Ausgabe.

HTTP kennt keinen eingebauten Mechanismus, mit dem Sie Informationen aus einer Seite so sichern können, dass Sie von anderen Seiten aus darauf zugreifen können. Das liegt daran, dass HTTP ein zustandsloses Protokoll ist. Die Rezepte 11.1, 11.3, 11.4 und 11.5 zeigen Ihnen verschiedene Möglichkeiten, wie Sie dieses fundamentale Problem umgehen können, um herauszufinden, welcher Benutzer Ihrer Website welche Anfragen tätigt.

Die Verarbeitung der vom Benutzer kommenden Daten ist das andere Hauptthema dieses Kapitels. Sie können den von einem Browser gelieferten Daten niemals vertrauen, daher ist es zwingend notwendig, stets alle Felder zu überprüfen – sogar verborgene Formularelemente. Die Validierung kann viele Formen haben, zum Beispiel behandelt Rezept 11.2, wie Sie sicherstellen, dass die Daten mit bestimmten Kriterien übereinstimmen, und Rezept 11.8, wie Sie Daten in HTML-Entities umwandeln, mit denen Sie Benutzereingaben sicher darstellen können. Des Weiteren zeigt Ihnen Rezept 11.7, wie Sie die Sicherheit Ihres Webservers schützen können, und Rezept 11.6 behandelt den Umgang mit von Benutzern hochgeladenen Dateien.

Immer wenn PHP eine Seite verarbeitet, wird diese auf GET- und POST-Formularvariablen, hochgeladene Dateien, anwendbare Cookies sowie Webserver- und Umgebungsvariablen untersucht. Auf diese kann dann in den folgenden Arrays direkt zugegriffen werden: `$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SERVER` und `$_ENV`. Sie enthalten alle Variablen, die durch GET-Anfragen, POST-Anfragen, hochgeladene Dateien, Cookies, den Webserver bzw. die Umgebung gesetzt worden sind. Außerdem gibt es noch `$_REQUEST`, ein einziges riesiges Array, das die Werte der anderen sechs Arrays enthält.

Wenn PHP die Elemente in `$_REQUEST` einträgt und dabei auf Arrays stößt, die einen Schlüssel mit dem gleichen Namen haben, greift es auf die Konfigurationsdirektive `variables_order` zurück. Standardmäßig ist `variables_order` auf EGPCS gesetzt (oder GPCS, wenn Sie die Konfigurationsdatei *php.ini-recommended* verwenden). Dementsprechend kopiert PHP erst die Umgebungsvariablen nach `$_REQUEST` und fügt danach die GET-, POST-, Cookie- und Webserver-Variablen (in dieser Reihenfolge) hinzu. Da beispielsweise in der Standardreihenfolge C nach P kommt, überschreibt ein Cookie namens `username` eine POST-Variable, die `username` heißt.

Wenn Sie keinen Zugang zu den PHP-Konfigurationsdateien haben, können Sie eine Einstellung mit `ini_get()` prüfen:

```
print ini_get('variables_order');  
EGPCS
```

Dies müssen Sie möglicherweise tun, weil Ihr ISP Ihnen nicht erlaubt, die Konfigurationseinstellungen anzusehen, oder weil Ihr Skript auf einem fremden Server läuft. Sie können die Einstellungen auch mit `phpinfo()` ansehen. Wenn Sie sich aber nicht darauf verlassen können, dass `variables_order` einen bestimmten Wert hat, sollten Sie direkt auf `$_GET` und `$_POST` anstelle von `$_REQUEST` zugreifen.

Die Arrays mit externen Variablen, wie etwa `$_REQUEST`, sind superglobal. Als solche müssen sie nicht innerhalb einer Funktion oder Klasse als `global` deklariert werden. Außerdem bedeutet dies, dass Sie diesen Variablen normalerweise nichts zuweisen sollten, da Sie sonst die in ihnen gespeicherten Daten überschreiben.

Vor PHP 4.1 existierten diese superglobalen Variablen noch nicht. Stattdessen gab es normale Arrays mit den Namen `$HTTP_COOKIE_VARS`, `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_POST_FILES` und `$HTTP_SERVER_VARS`. Diese Arrays sind aus Kompatibilitätsgründen immer noch verfügbar, aber mit den neuen Arrays kann man leichter arbeiten. Die alten Arrays werden nur gefüllt, wenn die Konfigurationsdirektive `track_vars` auf `on` geschaltet ist, allerdings ist seit PHP 4.0.3 diese Möglichkeit immer aktiviert.

Schließlich sind, sofern die Konfigurationsdirektive `register_globals` auf `on` steht, alle Variablen auch im globalen Namensraum verfügbar. Somit wird aus `$_GET['password']` einfach `$password`. Das ist zwar bequem, führt aber zu schweren Sicherheitsproblemen, da ein böswilliger Benutzer auf einfache Weise Variablen von außen setzen und damit vermeintlich sichere interne Variablen überschreiben kann. Seit PHP 4.2 ist `register_globals` daher standardmäßig `off`.

Wenn man dies alles weiß, kann man mit dem folgenden grundlegenden Skript alles zusammenbringen. Das Formular bittet den Benutzer, seinen Vornamen einzugeben, und antwortet mit einem Willkommensgruß. Der HTML-Code für das Formular sieht folgendermaßen aus:

```
<form action="/hello.php" method="post">  
Wie lautet Ihr Vorname?  
<input type="text" name="vorname">  
<input type="submit" value="Sag Hallo">  
</form>
```

Der Name des Texteingabeelements im Formular ist `first_name`. Außerdem ist `post` die Methode (method) für das Formular. Dies bedeutet, dass nach dem Absenden des Formulars `$_POST['first_name']` die Eingabe des Benutzers enthält. (Natürlich kann es auch leer sein, wenn nichts hineingeschrieben wurde.)

Der Einfachheit halber wollen wir aber annehmen, dass der Inhalt der Variablen gültig ist. (Was unter »gültig« zu verstehen ist, ist eine Definitionsfrage und kann von verschiedenen Kriterien abhängen, zum Beispiel davon, dass der Wert nicht leer ist, keinen Einbruchversuch in das System darstellt usw.) Somit wird hier die Fehlerprüfung außen vor gelassen, die zwar wichtig ist, bei diesem einfachen Beispiel aber eher stört. Hier ist als das einfache Skript *hello.php* zur Verarbeitung des Formulars:

```
echo 'Hallo ' . $_POST['vorname'] . '!' ;
```

Wenn der Benutzer den Vornamen Joe hat, gibt PHP aus:

```
Hallo Joe!
```

11.1 Formulareingaben verarbeiten

Problem

Sie möchten mit derselben HTML-Seite ein Formular ausgeben und danach die in diesem Formular eingegebenen Daten verarbeiten. Mit anderen Worten: Sie möchten die ungebremste Vermehrung von Seiten vermeiden, die jeweils einzelne Schritte einer Transaktion abarbeiten.

Lösung

Verwenden Sie ein verborgenes Feld im Formular, mit dem Sie Ihrem Programm mitteilen, dass es das Formular verarbeiten soll. In diesem Fall hat das verborgene Feld den Namen `stage` und den Inhalt `process`:

```
if (isset($_POST['stage']) && ('process' == $_POST['stage'])) {  
    process_form();  
} else {  
    print_form();  
}
```

Diskussion

Wenn Menschen in den frühen Tagen des Webs Formulare schufen, erstellten sie immer zwei Seiten: eine statische HTML-Seite mit dem Formular und ein Skript zur Verarbeitung des Formulars, das eine dynamisch generierte Antwort an den Benutzer lieferte. Dies war etwas unhandlich, denn *form.html* verwies auf *form.cgi*, und wenn die Menschen eine Seite änderten, mussten sie daran denken, auch die andere zu bearbeiten, da das Skript sonst vielleicht nicht mehr funktionsfähig war.

Formulare sind einfacher zu pflegen, wenn sich alle Teile in derselben Datei befinden und der Kontext bestimmt, welcher Teil gerade dargestellt werden soll. Verwenden Sie ein verborgenes Formularfeld mit dem Namen `stage` (Verarbeitungsstufe), mit dem Sie die Position im Verarbeitungsvorgang des Formulars verfolgen; es dient als Auslöser für die Schritte, die den jeweils passenden HTML-Code an den Benutzer zurückgeben. Manchmal ist es allerdings nicht möglich, das Programm auf diese Weise zu entwerfen, z.B. wenn Ihr Formular von einem Skript auf einem fremden Server verarbeitet wird.

Wenn Sie den HTML-Code für Ihr Formular schreiben, sollten Sie den Pfad zu Ihrer Seite in `action` nicht direkt fest einprogrammieren. Sonst können Sie Ihre Seite nicht umbenennen oder verlegen, ohne sie zu bearbeiten. PHP bietet eine hilfreiche Variable, die Sie stattdessen verwenden können:

```
$_SERVER['PHP_SELF']
```

Diese Variable ist ein Pseudonym für die URL der aktuellen Seite. Setzen Sie also den Wert des `action`-Attributs auf diese Variable, und Ihr Formular ruft sich immer selbst auf, auch wenn Sie die Datei an eine andere Stelle auf Ihrem Server verschoben haben.

Das Beispiel aus der Einführung zu diesem Kapitel sieht jetzt also folgendermaßen aus:

```
if (isset($_POST['stage']) && ('process' == $_POST['stage'])) {
    process_form();
} else {
    print_form();
}

function print_form() {
    echo <<<END
        <form action="$_SERVER[PHP_SELF]" method="post">
        Wie lautet Ihr Vorname?
        <input type="text" name="vorname">
        <input type="hidden" name="stage" value="process">
        <input type="submit" value="Sag Hallo">
        </form>
    END;
}

function process_form() {
    echo 'Hallo ' . $_POST['vorname'] . '!';
}
```

Wenn Ihr Formular mehr als einen Schritt erfordert, setzen Sie `stage` einfach für jeden Schritt auf einen neuen Wert.

Siehe auch

Rezept 11.3 zur Behandlung von Formularen mit mehreren Seiten.

11.2 Formulareingaben prüfen

Problem

Sie möchten sicherstellen, dass die durch ein Formular eingegebenen Daten bestimmten Kriterien genügen.

Lösung

Legen Sie eine Funktion an, die einen String zur Überprüfung annimmt und `true` zurückgibt, wenn der String der Prüfung standhält, und `false`, wenn dies nicht der Fall ist. In der Funktion verwenden Sie reguläre Ausdrücke und Vergleiche, um die Daten zu prüfen. Beispiel 11-1 zeigt als Beispiel die Funktion `pc_validate_zipcode()` zur Überprüfung einer amerikanischen Postleitzahl.

Beispiel 11-1: `pc_validate_zipcode()`

```
function pc_validate_zipcode($zipcode) {  
    return preg_match('/^[0-9]{5}([- ]?[0-9]{4})?$/ ', $zipcode);  
}
```

Und so wird sie verwendet:

```
if (pc_validate_zipcode($_REQUEST['zipcode'])) {  
    // Postleitzahl ist in Ordnung, Weiterverarbeitung möglich.  
    process_data();  
} else {  
    // Diese Postleitzahl ist fehlerhaft; Meldung ausgeben.  
    print "Eine US-Postleitzahl muss 5 Stellen haben ";  
    print "(oder 9 Stellen, wenn Sie ZIP+4 verwenden).";  
    print_form();  
}
```

Diskussion

Die Entscheidung darüber, was gültige und ungültige Daten ausmacht, ist fast eher eine philosophische Aufgabe als eine geradlinige Angelegenheit, bei der man einer Reihe festgelegter Schritte folgen kann. In vielen Fällen ist eine Lösung, die in einer Situation perfekt in Ordnung ist, in einer anderen Situation nicht korrekt.

Die einfachste Prüfung besteht darin sicherzustellen, dass ein Feld nicht leer ist. Diese Aufgabe kann man bestens mit der Funktion `empty()` erledigen:

```
if (empty($_POST['vorname'])) {  
    // Meldung: Feld VORNAME darf nicht leer sein  
}
```

Oftmals möchten Sie auch einfach nur sicherstellen, dass ein gesendeter Wert vom richtigen Datentyp ist und sich im richtigen Wertebereich befindet. Die Validierung einer

Altersangabe, bei der geprüft wird, ob der Wert größer oder gleich null ist, könnte wie folgt aussehen:

```
if (! ctype_digit($_POST['alter'])) {  
    print 'Altersangabe: Der Wert muss größer oder gleich null sein.';  
}
```

Die Validierung von Zahlen scheint auf den ersten Blick einfach zu sein, ist bei genauerer Betrachtung allerdings nicht ganz so leicht. Oftmals tendiert man dazu, `is_numeric()` zur Validierung von Zahlen zu verwenden. Allerdings ist das, was `is_numeric()` als Zahl ansieht, eher das, was ein Computer darunter versteht. Zum Beispiel werden hexadezimale Werte wie `0xCAFE` und Zahlen in exponentieller Schreibweise wie `10e40` als »numerisch« erkannt. Die `ctype`-Erweiterung bietet neben `ctype_digit()` noch viele weitere hilfreiche Funktionen, die unter <http://php.net/manual/ref ctype.php> beschrieben werden.

Danach kommen relativ einfache Prüfungen wie bei der US-Postleitzahl. Solche Probleme werden gewöhnlich mit ein oder zwei regulären Ausdrücken gelöst. Zum Beispiel findet:

```
 /^[0-9]{5}([- ]?[0-9]{4})?$/
```

alle gültigen amerikanischen Postleitzahlen.

Manchmal ist es allerdings schwierig, den richtigen regulären Ausdruck herauszufinden. Um sicherzustellen, dass jemand nur zwei Namen eingegeben hat, zum Beispiel »Alfred Aho«, können Sie mit diesem Ausdruck prüfen:

```
 /^[A-Za-z]+ +[A-Za-z]+$/
```

Tim O'Reilly besteht diesen Test jedoch nicht. Eine Alternative wäre `/^\\s+\\s+\\s+$/;`, aber dann würde Donald E. Knuth zurückgewiesen. Denken Sie also sorgfältig über die ganze Breite der gültigen Eingaben nach, bevor Sie Ihren regulären Ausdruck schreiben.

In manchen Fällen wird es auch mit regulären Ausdrücken schwierig nachzuprüfen, ob ein Feldinhalt zulässig ist. Eine besonders trickreiche Aufgabe ist die Prüfung einer E-Mail-Adresse, wie sie in Rezept 16.5 behandelt wird, und ebenso die Feststellung, ob jemand den Namen seines US-Bundesstaates korrekt angegeben hat. Sie können hier anhand einer Liste von Staaten prüfen, aber was soll geschehen, wenn die für die Post übliche Abkürzung angegeben wurde? Wird »MA« anstelle von Massachusetts funktionieren? Und was ist mit »Mass.«?

Als eine Möglichkeit, dieses Problem zu vermeiden, kann man dem Benutzer eine Dropdown-Liste von zuvor generierten Auswahlwerten präsentieren. Das `select`-Element zwingt die Benutzer durch das Formulardesign, den Staat immer in einem funktionierenden Format auszuwählen, und vermeidet damit Fehler. Dies bringt allerdings eine weitere Reihe von Schwierigkeiten mit sich. Was ist, wenn der Benutzer an einem Ort lebt, der keiner der angebotenen Wahlmöglichkeiten entspricht? Was tun, wenn die Auswahlliste so lang ist, dass dies keine realistische Lösung darstellt?

Es gibt verschiedene Möglichkeiten, derartige Problemen zu lösen. Erstens können Sie eine Option »Sonstige« in der Liste anbieten, sodass auch ein Nicht-US-Benutzer das Formular erfolgreich ausfüllen kann. (Andernfalls würde vermutlich ein zufälliger Wert ausgewählt, um die Site weiter verwenden zu können.) Als Nächstes können Sie den Anmeldevorgang in eine zweiteilige Folge aufteilen. Bei einer langen Liste von Optionen beginnt der Benutzer zuerst damit, einen Buchstaben aus dem Alphabet auszuwählen, mit dem seine Auswahl beginnt, und eine neue Seite präsentiert ihm dann eine Liste mit den Auswahlmöglichkeiten, die mit diesem Buchstaben beginnen.

Wie können Sie sicherstellen, dass bei Verwendung eines Drop-downs auch tatsächlich ein Wert ausgewählt wird, der im Drop-down angeboten wird? Der Nutzer kann einen Request ja auch manipulieren und einen Wert senden, der gar nicht zur Auswahl stand. Die Lösung dafür ist die Verwendung der Funktion `in_array()`:

```
$auswahl = array('eier' => 'Spiegelei',
                'toast' => 'Buttertoast mit Marmelade',
                'kaffee' => 'Heißer Kaffee');
echo "<select name='essensauswahl'>\n";
foreach ($auswahl as $key => $name) {
    echo "<option value='$key'>$name</option>\n";
}
echo "</select>";
// Später die Validierung
if (! array_key_exists($_POST['essensauswahl'], $choices)) {
    echo "Sie müssen eine valide Auswahl aus der Liste treffen.";
}
```

Die Auswahlmöglichkeiten müssen in einem Array vorliegen, aus dem auch gleich die Auswahlliste für das Drop-down generiert werden kann. Wenn der vom Benutzer gewählte Wert vom Browser gesendet wird, können Sie mit `in_array()` sicherstellen, dass es sich um eine valide Eingabe handelt, die vom Drop-down angeboten wurde.

Schließlich gibt es sogar noch diffizilere Probleme. Was tun Sie, wenn Sie zwar sicherstellen möchten, dass ein Benutzer eine Information korrekt eingegeben hat, ihm aber nicht mitteilen wollen, dass Sie dies überprüft haben? Eine Situation, bei der dies wichtig ist, ist ein Preisausschreiben; hier gibt es häufig eine spezielle Code-Box auf dem Eingabeformular, in der die Benutzer eine Zeichenkette – AD78DQ – aus einer zuvor erhaltenen E-Mail oder einem Werbezettel eintragen sollen. Sie möchten sicherstellen, dass es keine Tippfehler gibt, oder den jeweiligen Benutzer andernfalls nicht als gültigen Teilnehmer zählen. Außerdem möchten Sie ihm nicht ermöglichen, Codes einfach zu erraten, da man dann die Codes ausprobieren und auf diese Weise das System knacken könnte.

Die Lösung besteht darin, zwei Eingabefelder anzubieten. Der Benutzer gibt seinen Code zweimal ein; wenn die beiden Felder übereinstimmen, akzeptieren Sie die Daten als legal und überprüfen sie dann (im Stillen). Wenn die Felder nicht übereinstimmen, weisen Sie die Eingabe zurück und lassen sie vom Benutzer korrigieren. Dieses Vorgehen vermeidet Tippfehler, legt aber nicht offen, wie die Code-Überprüfung funktioniert; so können auch falsch geschriebene E-Mail-Adressen vermieden werden.

Letztendlich ermöglicht PHP nur die Validierung auf dem Server. Serverseitige Validierung erfordert, dass eine Anfrage an den Server gestellt wird und eine Seite als Antwort zurückgesandt wird; im Endeffekt kann dies langsam sein. Daneben gibt es die Möglichkeit der clientseitigen Validierung mithilfe von JavaScript. Während clientseitige Eingabeüberprüfung schneller ist, legt sie aber Ihren Code den Anwendern gegenüber offen und funktioniert möglicherweise nicht, wenn ein Client JavaScript nicht unterstützt. Deshalb sollten Sie clientseitigen Validierungs-Code immer auf dem Server duplizieren.

Siehe auch

Rezept 16.5 zu einem regulären Ausdruck zur Prüfung von E-Mail-Adressen; Kapitel 9, »Validierung mit PHP und JavaScript« in *Webdatenbank-Applikationen mit PHP & MySQL*, 2. Auflage (Hugh Williams und David Lane, O'Reilly Verlag).

11.3 Mit mehrseitigen Formularen arbeiten

Problem

Sie möchten ein Formular verwenden, das mehr als eine Seite anzeigt und die Daten von einer zur nächsten Seite aufhebt.

Lösung

Machen Sie vom Session-Tracking Gebrauch:

```
session_start();
$_SESSION['username'] = $_GET['username'];
```

Sie können auch Variablen aus den vorhergehenden Seiten eines Formulars als verborgene Eingabefelder in späteren Seiten einfügen:

```
<input type="hidden" name="username"
value="<?php echo htmlentities($_GET['username']); ?>">
```

Diskussion

Benutzen Sie möglichst immer das Session-Tracking. Dies ist sicherer, da Benutzer die Session-Variablen nicht verändern können. Um eine Sitzung zu beginnen, rufen Sie `session_start()` auf; damit wird eine neue Session angelegt oder eine vorhandene fortgeführt. Beachten Sie, dass dieser Schritt nicht erforderlich ist, wenn Sie in Ihrer `php.ini`-Datei `session.auto_start` eingeschaltet haben. Variablen, die `$_SESSION` zugewiesen sind, werden automatisch propagiert. Im Lösungsbeispiel wird die Formularvariable `username` dadurch bewahrt, dass sie mit `$_GET['username']` mit `$_SESSION['username']` der Session zugewiesen wird.

Um während einer nachfolgenden Anfrage auf diesen Wert zuzugreifen, rufen Sie `session_start()` auf und prüfen dann `$_SESSION['username']`:

```

session_start();
$username = htmlentities($_SESSION['username']);
print "Hallo $username.";

```

Wenn Sie in diesem Fall `session_start()` nicht aufrufen, ist `$_SESSION` nicht gesetzt.

Achten Sie darauf, dass der Server und der Speicherort Ihrer Session-Dateien (Dateisystem, Datenbank usw.) gesichert sind; andernfalls ist Ihr System verwundbar, wenn falsche Identitäten vorgetäuscht werden.

Wenn in Ihrer PHP-Installation das Session-Tracking nicht aktiviert ist, können Sie ersatzweise auch verborgene Formularvariablen verwenden. Die Weitergabe von Daten mithilfe verborgener Formularelemente ist aber nicht sicher, da jedermann diese Felder editieren und somit Anfragen verfälschen kann. Allerdings können Sie mit geringem Aufwand die Sicherheit auf ein verlässliches Niveau anheben.

Am einfachsten benutzen Sie verborgene Felder, indem Sie diese in Ihr Formular einfügen.

```

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="get">

<input type="hidden" name="username"
value="<?php echo htmlentities($_GET['username']); ?>">

```

Wenn dieses Formular erneut abgesendet wird, enthält `$_GET['username']` seinen vorherigen Wert, sofern nicht jemand ihn verändert hat.

Eine kompliziertere, aber auch sicherere Lösung besteht darin, dass Sie Ihre Variablen mithilfe von `serialize()` in einen String konvertieren, einen geheimen Hash von den Daten berechnen und diese beiden Teile der Information in das Formular einfügen. Bei der nächsten Anfrage validieren Sie dann die Daten und deserialisieren sie. Wenn sie den Verifikationstest nicht bestehen, wissen Sie, dass irgendjemand die Informationen verändert hat.

Die in Beispiel 11-2 dargestellte Codierungsfunktion `pc_encode()` übernimmt die zu codierenden Daten in Form eines Arrays.

Beispiel 11-2: `pc_encode()`

```

$secret = 'Foo25bAr52baZ';

function pc_encode($data) {
    $data = serialize($data);
    $hash = md5($GLOBALS['secret'] . $data);
    return array($data, $hash);
}

```

Die Funktion `pc_encode()` serialisiert die Daten zu einem String, berechnet dann einen Validierungs-Hash und gibt diese beiden Variablen zurück.

Die Funktion `pc_decode()` in Beispiel 11-3 macht die Arbeit ihres Gegenstücks wieder rückgängig.

Beispiel 11-3: `pc_decode()`

```
function pc_decode($data, $hash) {
    if (!empty($data) && !empty($hash)) {
        if (md5($GLOBALS['secret'] . $data) == $hash) {
            return unserialize($data);
        } else {
            error_log("Validierungsfehler: Daten wurden verändert");
            return false;
        }
    }
    return false;
}
```

Die Funktion `pc_decode()` erzeugt den Hash mit dem geheimen Wort erneut und vergleicht ihn mit dem Hash-Wert aus dem Formular. Wenn sie gleich sind, ist `$data` gültig und kann deserialisiert werden. Wenn der Test fehlschlägt, schreibt die Funktion eine Mitteilung in das Fehlerprotokoll und gibt `false` zurück.

Diese Funktionen arbeiten folgendermaßen zusammen:

```
<?php
$secret = 'Foo25bAr52baZ';

// Alte Daten laden und validieren.
if (! $data = pc_decode($_GET['data'], $_GET['hash'])) {
    // Einbruchsversuch
}

// Formular verarbeiten (neue Formulare Daten sind in $_GET).

// $data aktualisieren.
$data['username'] = $_GET['username'];
$data['stage']++;
unset($data['password']);

// Ergebnis codieren.
list ($data, $hash) = pc_encode($data);

// Daten und Hash in das Formular einfügen.
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="get">
...
<input type="hidden" name="data"
    value="<?php echo htmlentities($data); ?>">
<input type="hidden" name="hash"
    value="<?php echo htmlentities($hash); ?>">
</form>
```

Am Anfang des Skripts werden die Variablen aus dem Formular zur Decodierung an `pc_decode()` übergeben. Nachdem die Informationen in `$data` geladen sind, kann die Formularverarbeitung fortgesetzt werden, indem `$_GET` auf neue Variablen und `$data` auf vorhandene Daten geprüft wird. Wenn Sie damit fertig sind, aktualisieren Sie `$data`, sodass es auch die neuen Variablen enthält, codieren den Inhalt und berechnen dabei einen neuen Hash. Schließlich geben Sie das neue Formular aus, dabei fügen Sie `$data` und `$hash` als verborgene Variablen ein.

Siehe auch

Rezept 10.5 und 10.6 mit Informationen zur Verwendung des Session-Moduls; Rezept 11.8 mit Einzelheiten zur Verwendung von `htmlentities()` zum Auszeichnen von Kontrollzeichen in der HTML-Ausgabe; Rezept 17.3 mit Informationen zum Verifizieren von Daten mit Hash-Codes; die Dokumentation zum Session-Tracking unter <http://www.php.net/session> und in Rezept 10.5; die Dokumentationen zu `serialize()` unter <http://www.php.net/serialize> und `unserialize()` unter <http://www.php.net/unserialize>.

11.4 Formulare mit erhaltenen Informationen und Fehlermeldungen erneut anzeigen

Problem

Wenn es ein Problem mit den in einem Formular eingegebenen Daten gibt, möchten Sie anstelle einer generischen Fehlermeldung am Anfang der Seite die Fehlermeldungen in der Nähe der Problemfelder anzeigen. Außerdem möchten Sie, dass die Informationen erhalten bleiben, die der Benutzer beim ersten Mal in das Formular eingegeben hat.

Lösung

Verwenden Sie ein Array `$errors` und speichern Sie Ihre Meldungen in dem Array, wobei Sie den Feldnamen als Index benutzen.

```
if (! pc_validate_zipcode($_REQUEST['zipcode'])) {
    $errors['zipcode'] = "Dies ist eine fehlerhafte Postleitzahl. "
        . "Postleitzahlen müssen aus 5 Ziffern "
        . "bestehen und dürfen keine Buchstaben "
        . "enthalten.";
}
```

Wenn Sie dann das Formular erneut ausgeben, können Sie jeden Fehler bei dem zugehörigen Feld anzeigen und den ursprünglichen Wert in das Feld einfügen:

```
echo $errors['zipcode'];
$value = isset($_REQUEST['zipcode']) ?
    htmlentities($_REQUEST['zipcode']) : '';
echo "<input type='text' name='zipcode' value='\"$value\">";
```

Diskussion

Wenn Ihre Benutzer beim Ausfüllen eines langen Formulars auf Fehler stoßen, können Sie die Bedienbarkeit Ihres Formulars insgesamt dadurch verbessern, dass Sie auf die Stelle hinweisen, an der genau die Fehler korrigiert werden müssen.

Alle Fehler in einem einzigen Array zusammenzuführen hat viele Vorteile. Erstens können Sie einfach prüfen, ob bei Ihrem Prüfungsvorgang Fehler gefunden wurden, die korrigiert werden müssen; verwenden Sie dazu einfach `count($errors)`. Diese Methode ist einfacher, als dafür eine gesonderte Variable zu benutzen, insbesondere wenn der Ablauf komplex oder über mehrere Funktionen verteilt ist. Beispiel 11-4 zeigt die Validierungsfunktion `pc_validate_form()`, die ein `$errors`-Array verwendet.

Beispiel 11-4: pc_validate_form()

```
function pc_validate_form() {
    if (! pc_validate_zipcode($_POST['zipcode'])) {
        $errors['zipcode'] = "Eine Postleitzahl hat 5 Ziffern";
    }

    if (! pc_validate_email($_POST['email'])) {
        $errors['email'] = "E-Mail-Adressen sehen aus wie user@example.com";
    }

    return $errors;
}
```

Dies ist sauberer Code, denn alle Fehler werden in einer Variablen gespeichert. Diese Variable können Sie einfach herumreichen, wenn Sie sie nicht im globalen Geltungsbereich unterbringen möchten.

Dadurch dass Sie den Variablennamen als Schlüssel verwenden, erhalten Sie die Verbindung zwischen dem Feld, das den Fehler verursacht hat, und der eigentlichen Fehlermeldung. Diese Verknüpfungen erleichtern es auch, die Elemente in einer Schleife zu durchlaufen, wenn Sie die Fehler anzeigen.

Die wiederholte Darstellung des Formulars können Sie automatisieren; die Funktion `pc_print_form()` in Beispiel 11-5 zeigt Ihnen, wie das geht.

Beispiel 11-5: pc_print_form()

```
function pc_print_form($errors) {
    $fields = array('name' => 'Name',
                   'rank' => 'Rank',
                   'serial' => 'Serial');

    if (count($errors)) {
        echo 'Bitte korrigieren Sie die Fehler im folgenden Formular.';
    }
}
```

Beispiel 11-5: `pc_print_form()` (Fortsetzung)

```

echo '<table>';

// Fehlermeldungen und Formularvariablen ausgeben.
foreach ($fields as $field => $field_name) {
    // Zeile beginnen.
    echo '<tr><td>';

    // Fehler ausgeben.
    if (!empty($errors[$field])) {
        echo $errors[$field];
    } else {
        echo '&nbsp;'; // vermeidet unschöne Tabellen
    }

    echo "</td><td>";

    // Name und Eingabefeld darstellen.
    $value = isset($_REQUEST[$field]) ?
        htmlentities($_REQUEST[$field]) : '';

    echo "$field_name: ";
    echo "<input type=\"text\" name=\"$field\" value=\"$value\">";
    echo '</td></tr>';
}

echo '</table>';
}

```

Der komplexe Teil in `pc_print_form()` resultiert aus dem `$fields`-Array, dessen Schlüssel die Variablennamen und dessen Werte der anzuzeigenden Feldnamen sind. Wenn Sie das Array zu Beginn der Funktion definieren, können Sie eine Schleife erzeugen und mit `foreach` die Werte durchlaufen; andernfalls würden Sie drei getrennte Zeilen identischen Codes benötigen. Dies passt zu dem Variablennamen als Schlüssel in `$errors`, denn Sie können die Fehlermeldung in der Schleife einfach herausfinden, indem Sie `$errors[$field]` überprüfen.

Wenn Sie dieses Beispiel über `input`-Felder vom Typ `text` hinaus erweitern möchten, modifizieren Sie `$fields` so, dass es weitere Metainformationen über Ihre Formularfelder enthält:

```

$fields = array('name' => array('name' => 'Name', 'type' => 'text'),
               'rank' => array('name' => 'Rank', 'type' => 'password'),
               'serial' => array('name' => 'Serial', 'type' => 'hidden')
);

```

Siehe auch

Rezept 11.2 zur einfachen Formularvalidierung.

11.5 Mehrfaches Absenden desselben Formulars verhindern

Problem

Sie möchten dafür sorgen, dass Anwender nicht mehrmals dasselbe Formular übersenden.

Lösung

Sie generieren einen eindeutigen Identifikator und speichern das Kennzeichen im Formular als verborgenes Feld. Bevor Sie das Formular verarbeiten, prüfen Sie, ob das Kennzeichen bereits übersandt wurde. Wenn dies nicht der Fall ist, fahren Sie fort; wurde es bereits übersandt, sollten Sie einen Fehler generieren.

Beim Erzeugen des Formulars erhalten Sie mithilfe von `uniqid()` einen eindeutigen Identifikator:

```
<?php
$unique_id = uniqid(microtime(),1);
...
?>
<input type="hidden" name="unique_id" value="<?php echo $unique_id; ?>">
</form>
```

Beim Verarbeiten achten Sie dann auf diese ID:

```
$unique_id = $dbh->quote($_GET['unique_id']);
$stmt = $dbh->query("SELECT * FROM database WHERE unique_id = $unique_id");

if ($stmt->numRows()) {
    // Bereits erhalten; Fehler auslösen.
} else {
    // Mit den Daten weiterarbeiten.
}
```

Diskussion

Aus verschiedenen Gründen senden Benutzer ein Formular häufig mehrmals ab. Meistens ist es ein Maus-Lapsus: ein Doppelklick auf den Absenden-Button. Benutzer drücken auch schon mal die Zurück-Schaltfläche des Browsers, um die Informationen noch einmal zu sehen oder sie zu ändern, klicken dann aber erneut auf »Absenden« anstelle von »Vorwärts«. Es kann auch absichtlich sein, beispielsweise beim Versuch, die Wahlurne einer Online-Befragung oder den Briefkasten eines Preisausschreibens vollzustopfen. Unsere Lösung verhindert versehentliche Attacken und kann böswillige Anwender zumindest behindern. Sie kann allerdings nicht jede betrügerische Benutzung ausschalten – dazu sind kompliziertere Maßnahmen erforderlich.

Die Lösung verhindert, dass Ihre Datenbank mit allzu vielen Kopien desselben Datensatzes überhäuft wird. Indem Sie ein Kennzeichen generieren und in das Formular einfügen, können Sie dieses spezielle Exemplar eindeutig identifizieren, sogar wenn Cookies deaktiviert sind. Wenn Sie die Daten des Formulars speichern, legen Sie das Kennzeichen mit ihnen ab. Auf diese Weise können Sie einfach prüfen, ob Sie dieses Formular und den zugehörigen Satz in der Datenbank bereits gesehen haben.

Beginnen Sie, indem Sie Ihrer Datenbanktabelle eine zusätzliche Spalte namens `unique_id` hinzufügen, die den Identifikator aufnehmen soll. Wenn Sie die Daten für einen Satz einfügen, fügen Sie auch die ID hinzu, zum Beispiel:

```
$username = $dbh->quote($_GET['username']);  
$unique_id = $dbh->quote($_GET['unique_id']);  
  
$sth = $dbh->query("INSERT INTO members ( username, unique_id)  
VALUES ($username, $unique_id)");
```

Wenn Sie die genaue Datenbankzeile mit dem Formular verknüpfen, können Sie mit einer erneuten Übersendung besser umgehen. Auf die Frage, was dann zu tun ist, kann es hier keine korrekte Antwort geben; es hängt von der jeweiligen Situation ab. In manchen Fällen können Sie die zweite Zusendung insgesamt ignorieren. In anderen werden Sie prüfen wollen, ob sich der Datensatz geändert hat, und wenn dies zutrifft, dem Benutzer eine Dialogbox mit der Frage präsentieren, ob die Daten mit der neuen Information überschrieben werden oder die alten Daten unverändert bleiben sollen. Schließlich können Sie auf die zweite Übersendung des Formulars auch so reagieren, dass Sie den Datensatz stillschweigend überschreiben, und der Benutzer erfährt niemals etwas von dem Problem.

Alle diese Möglichkeiten sollten unter Berücksichtigung der Besonderheiten der jeweiligen Interaktion abgewogen werden. Unserer Auffassung nach gibt es keinen Grund, das Benutzererlebnis durch die Defizite des HTTP-Protokolls einschränken zu lassen. Daher ist die dritte Option, die stillschweigende Aktualisierung, zwar nicht das, was normalerweise geschieht, aber in vieler Hinsicht die natürlichste Wahl. Anwendungen, die mit diesem Vorgehen entwickelt werden, sind benutzerfreundlicher, während die beiden anderen Methoden die meisten Anwender frustrieren oder verwirren.

Vielleicht neigen Sie dazu, die Generierung eines zufälligen Kennzeichens zu vermeiden und stattdessen die um eins erhöhte Anzahl der bereits in der Datenbank vorhandenen Datensätze zu verwenden? Dann ist das Kennzeichen identisch mit dem Primärschlüssel, und Sie benötigen keine zusätzliche Spalte. Bei diesem Vorgehen gibt es aber (mindestens) zwei Probleme. Erstens führt dies zu einer Race Condition. Was geschieht, wenn eine zweite Person das Formular aufruft, bevor die erste Person es abgeschlossen hat? Dann erhält das zweite Formular dasselbe Kennzeichen wie das erste, und es kommt zu einem Konflikt. Diesen können Sie umgehen, indem Sie beim Abruf des Formulars eine leere Zeile in die Datenbank einfügen, sodass die zweite Personen eine um eins höhere

Zahl als die erste erhält. Dies kann jedoch zu leeren Zeilen in der Datenbank führen, wenn Benutzer das Formular nicht abschließen.

Außerdem spricht dagegen, dass es dann trivial ist, einen anderen Satz in der Datenbank zu ändern, weil man einfach die ID auf eine andere Zahl setzen kann. Abhängig von Ihren Sicherheitseinstellungen können mit einer gefälschten GET- oder POST-Sendung die Daten problemlos geändert werden. Ein langes, zufällig gebildetes Kennzeichen kann dagegen nicht einfach dadurch erraten werden, dass man eine andere Ganzzahl aussucht.

Siehe auch

Rezept 17.3 mit weiteren Einzelheiten zur Verifikation von Daten mit Hash-Codes; die Dokumentation zu `uniqid()` unter <http://www.php.net/uniqid>.

11.6 Hochgeladene Dateien verarbeiten

Problem

Sie möchten eine vom Benutzer hochgeladene Datei verarbeiten.

Lösung

Verwenden Sie das Array `$_FILES`:

```
// Aus <input name="event" type="file">
if (is_uploaded_file($_FILES['event']['tmp_name'])) {
    // Datei auf dem Bildschirm ausgeben.
    readfile($_FILES['event']['tmp_name']);
}
```

Diskussion

Seit PHP 4.1 erscheinen alle hochgeladenen Dateien in dem superglobalen Array `$_FILES`. Zu jeder Datei gibt es fünf Informationen:

`name`

Der dem Eingabeelement im Formular zugewiesene Name.

`type`

Der MIME-Typ der Datei.

`size`

Die Größe der Datei in Bytes.

`tmp_name`

Der Ort, an dem die Datei temporär auf dem Server gespeichert ist.

error

Ein Fehlercode, der bei fehlgeschlagenem Upload genauere Problemhinweise gibt (verfügbar in PHP-Versionen ≥ 4.2).

Wenn Sie eine frühere PHP-Version einsetzen, müssen Sie stattdessen `$HTTP_POST_FILES` verwenden.

Mögliche Werte des `error`-Elements sind:

`UPLOAD_ERR_OK` (0)

Upload erfolgreich (keine Fehler).

`UPLOAD_ERR_INI_SIZE` (1)

Die Größe der hochgeladenen Datei ist größer als der Wert der `upload_max_filesize-php.ini`-Direktive.

`UPLOAD_ERR_FORM_SIZE` (2)

Die Größe der hochgeladenen Datei ist größer als der Wert des Formularelements `MAX_FILE_SIZE`.

`UPLOAD_ERR_PARTIAL` (3)

Die Datei wurde nur teilweise hochgeladen.

`UPLOAD_ERR_NO_FILE` (4)

Es wurde keine Datei hochgeladen.

`UPLOAD_ERR_NO_TMP_DIR` (6)

Der Upload ist fehlgeschlagen, weil kein temporäres Verzeichnis zum Abspeichern vorhanden war (verfügbar in PHP 4.3.10, 5.0.3 und späteren Versionen).

`UPLOAD_ERR_CANT_WRITE` (7)

PHP konnte die Datei nicht auf den Datenträger schreiben (verfügbar in PHP $\geq 5.1.0$).

Die aufgeführten Konstanten sind ab PHP 4.3 oder später verfügbar. In früheren Versionen benutzen Sie bitte die in Klammern aufgeführte Zahl zum Auswerten des Werts für den Fehlercode.

Nachdem Sie eine Datei aus dem Array ausgewählt haben, prüfen Sie mit `is_uploaded_file()`, ob es sich bei der Datei, die Sie gerade verarbeiten möchten, um eine legitime, von einem Benutzer hochgeladene Datei handelt. Dann verarbeiten Sie die Datei, wie Sie jede andere Datei auf Ihrem System verarbeiten. Führen Sie die Prüfung in jedem Fall durch; wenn Sie dem vom Benutzer übergebenen Dateinamen blind vertrauen, kann jemand die Anfrage verändern und solche Namen wie `/etc/passwd` in die Liste der zu verarbeitenden Dateien einfügen.

Sie können auch die Datei an einen permanenten Speicherort verschieben; verwenden Sie für den sicheren Transfer der Datei die Funktion `move_uploaded_file()`:

```
// Die Datei verschieben: move_uploaded_file() prüft zugleich die
// Legitimität der Datei, daher brauchen Sie nicht is_uploaded_file()
// aufzurufen.
move_uploaded_file($_FILES['event']['tmp_name'], '/path/to/file.txt');
```

Beachten Sie, dass der in `tmp_name` gespeicherte Name der komplette Dateipfad ist und nicht nur der Basisname. Mithilfe von `basename()` können Sie bei Bedarf die davor stehenden Verzeichnisnamen abschneiden.

Stellen Sie sicher, dass PHP das Lese- und Schreibrecht sowohl in dem Verzeichnis hat, in dem die temporären Dateien gespeichert werden (in der Konfigurationsdirektive `upload_tmp_dir` können Sie nachsehen, wo dies ist), als auch an dem Ort, an den Sie die Datei hinkopieren möchten. Dies kann häufig der Benutzer `nobody` oder `apache` anstelle Ihres persönlichen Benutzernamens sein. Aus diesem Grund werden Sie, wenn Sie im `safe_mode` laufen, nach dem Kopieren der Datei an einen anderen Ort wahrscheinlich nicht noch einmal darauf zugreifen können.

Die Verarbeitung von Dateien ist häufig eine subtile Aufgabe, da nicht alle Browser die gleichen Informationen übersenden. Es ist aber wichtig, dass es auf korrekte Weise geschieht, da Sie sonst ein mögliches Sicherheitsloch öffnen. Letztendlich ermöglichen Sie jedem Fremden, eine beliebige von ihm gewählte Datei auf Ihren Rechner zu laden; böswillige Menschen können darin eine Gelegenheit sehen, in den Rechner einzudringen oder ihn zum Absturz zu bringen.

Aus diesem Grund bietet PHP eine Anzahl von Möglichkeiten, mit denen Sie Restriktionen auf hochgeladene Dateien legen können; zum Beispiel kann das Dateihochladen insgesamt abgeschaltet werden. Wenn Sie bei der Verarbeitung von hochgeladenen Dateien auf Schwierigkeiten stoßen, sollten Sie nachsehen, ob Ihre Datei nicht vielleicht abgewiesen wird, weil sie ein Sicherheitsrisiko darzustellen scheint.

Zu diesem Zweck stellen Sie zuerst sicher, dass `file_uploads` in der Konfigurationsdatei auf `On` gestellt ist. Danach prüfen Sie, ob die Datei nicht größer als `upload_max_filesize` ist; die Vorgabe von 2 MByte soll verhindern, dass jemand durch Auffüllen der Festplatte mit einer gigantischen Datei den Rechner zum Absturz bringen kann. Zusätzlich gibt es noch eine Direktive `post_max_size`, mit der die Maximalgröße aller POST-Daten innerhalb einer einzelnen Anfrage festgelegt wird; der Anfangswert hierfür beträgt 8 MByte.

Wenn es Ihnen nicht gelingt, `$_FILES` mit Informationen zu füllen, sollten Sie auch in Bezug auf Browser-Unterschiede und Benutzerfehler sicherstellen, dass Sie in das öffnende Tag des Formulars `enctype="multipart/form-data"` eintragen. PHP benötigt dies, um die Verarbeitung auszulösen. Wenn Sie dies nicht tun können, müssen Sie `$HTTP_RAW_POST_DATA` manuell parsen. (Siehe die RFCs 1521 und 1522 bezüglich der MIME-Spezifikation unter <http://www.faqs.org/rfcs/rfc1521.html> und <http://www.faqs.org/rfcs/rfc1522.html>.)

Außerdem setzen PHP-Versionen vor 4.1 `tmp_name` auf `none`, wenn keine Datei zum Hochladen ausgewählt worden ist; neuere Versionen setzen `tmp_name` dann auf einen Leer-String. PHP 4.2.1 lässt Dateien mit der Länge 0 zu. Um sicher zu sein, dass eine Datei hochgeladen wurde und nicht leer ist (obwohl unter Umständen leere Dateien das sind, was Sie benötigen), müssen Sie sicherstellen, dass `tmp_name` gesetzt und `size` größer als 0 ist. Und schließlich senden nicht alle Browser denselben MIME-Typ für eine Datei; was sie senden, hängt von ihrer jeweiligen Kenntnis verschiedener Dateierarten ab.

Siehe auch

Die Dokumentation zum Umgang mit hochgeladenen Dateien unter <http://www.php.net/features.file-upload> und zu `basename()` unter <http://www.php.net/basename>.

11.7 Die Formularverarbeitung durch PHP absichern

Problem

Sie möchten Eingabevariablen aus Formularen sicher verarbeiten und niemandem die böswillige Veränderung Ihres Codes ermöglichen.

Lösung

Deaktivieren Sie die Konfigurationsdirektive `register_globals` und greifen Sie nur über das `$_REQUEST`-Array auf Variablen zu. Noch weiter erhöhen Sie die Sicherheit, indem Sie `$_GET`, `$_POST` und `$_COOKIE` verwenden und somit genau wissen, wo Ihre Variablen herkommen.

Stellen Sie zu diesem Zweck sicher, dass in Ihrer `php.ini`-Datei die folgende Zeile auftaucht:

```
register_globals = Off
```

Seit PHP 4.2 ist dies die standardmäßige Konfiguration.

Diskussion

Wenn `register_globals` eingeschaltet ist, werden externe Variablen, darunter auch die aus Formularen und Cookies, direkt in den globalen Namensraum importiert. Dies ist zwar sehr bequem, kann aber auch einige Sicherheitslücken mit sich bringen, wenn Sie nicht konsequent die Inhalte und die Herkunft Ihrer Variablen prüfen. Warum ist das so? Es kann vorkommen, dass eine intern verwendete Variable, die nicht für den Zugriff von außen vorgesehen ist, ohne Ihr Wissen von außen überschrieben werden kann.

Dazu ein einfaches Beispiel: Sie haben eine Seite, in der Anwender ihren Benutzernamen und ihr Passwort eingeben. Wenn beide validiert sind, geben Sie eine Identifikationsnummer zurück; anhand dieses numerischen Identifikators suchen Sie die persönlichen Informationen des Benutzers und geben sie aus:

```
// Voraussetzung: magic_quotes_gpc ist auf Off gesetzt.  
$username = $dbh->quote($_GET['username']);  
$password = $dbh->quote($_GET['password']);  
  
$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND  
password = $password");
```

```

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
} else {
    "Benutzername und Passwort ungültig";
}

if (!empty($id)) {
    $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
}

```

Normalerweise wird `$id` nur als Ergebnis der Datenbanksuche zur Verifikation durch Ihr Programm gesetzt. Wenn allerdings jemand den GET-String verändert und einen Wert für `$id` übergibt, führt Ihr Skript, sofern `register_globals` aktiviert ist, auch nach einer erfolglosen Suche nach Benutzernamen und Passwort die zweite Datenbankabfrage durch und gibt die Ergebnisse zurück. Ohne `register_globals` bleibt `$id` leer, da nur `$_REQUEST['id']` (und `$_GET['id']`) gesetzt werden.

Es gibt natürlich noch andere Möglichkeiten, dieses Problem auch mit `register_globals` zu lösen. Sie können etwa Ihren Code so umstrukturieren, dass diese Lücke nicht mehr entsteht:

```

$sth = $dbh->query("SELECT id FROM users WHERE username = $username AND
password = $password");

if (1 == $sth->numRows()) {
    $row = $sth->fetchRow(DB_FETCHMODE_OBJECT);
    $id = $row->id;
    if (!empty($id)) {
        $sth = $dbh->query("SELECT * FROM profile WHERE id = $id");
    }
} else {
    "Benutzername und Passwort ungültig";
}

```

Nun verwenden Sie `$id` nur noch, wenn der Wert explizit durch einen Datenbankaufruf gesetzt worden ist. Manchmal ist so etwas aber schwierig, wenn das Programm nicht entsprechend ausgelegt ist. Alternativ können Sie auch am Anfang des Skripts alle Variablen mit `unset()` löschen oder sie initialisieren:

```
unset($id);
```

Dadurch wird ein fehlerhafter `$id`-Wert entfernt, bevor er Ihr Programm beeinflussen kann. Da PHP aber keine Variablen-Initialisierung verlangt, kann man dies unter Umständen einmal vergessen, und so kann sich der Fehler einschleichen, ohne dass es eine Warnung von PHP gibt.

Siehe auch

Die Dokumentation zu `register_globals` unter <http://www.php.net/manual/en/security.globals.php>.

11.8 Steuerzeichen in Benutzerdaten durch Escape-Sequenzen ersetzen

Problem

Sie möchten Daten, die von Benutzern eingegeben worden sind, auf einer HTML-Seite sicher ausgeben, um z.B. Cross-Site Scripting (XSS) Angriffen vorzubeugen.

Lösung

Für HTML-Code, den Sie als einfachen Text ausgeben möchten und der Links und andere Tags enthält, verwenden Sie `htmlentities()`:

```
echo htmlentities('<p>O'Reilly & Associates</p>');  
&lt;p&gt;O'Reilly & Associates&lt;/p&gt;
```

Diskussion

PHP verfügt über einige Funktionen, mit denen Sie Zeichen durch HTML-Entities ersetzen können. Die elementarste von ihnen ist `htmlspecialchars()`, die folgende vier Zeichen ersetzt: `<` `>` `"` und `&`. Abhängig von optionalen Parametern kann sie auch `'` anstelle von oder zusätzlich zu `"` übersetzen. Für komplexere Codierungen verwenden Sie `htmlentities()`; diese Funktion erweitert `htmlspecialchars()`, sodass alle Zeichen ersetzt werden, zu denen es HTML-Entities gibt.

```
$html = "<a href='fletch.html'>Hans' Lieblingsfilm.</a>\n";  
print htmlspecialchars($html); // doppelte Anführungszeichen  
print htmlspecialchars($html, ENT_QUOTES); // einfache und doppelte Anführungszeichen  
print htmlspecialchars($html, ENT_NOQUOTES); // weder noch  
&lt;a href="&quot;fletch.html&quot;&gt;Hans' Lieblingsfilm.&lt;/a&gt;  
&lt;a href="&quot;fletch.html&quot;&gt;Hans&#039; Lieblingsfilm.&lt;/a&gt;  
&lt;a href="fletch.html"&gt;Hans' Lieblingsfilm.&lt;/a&gt;
```

Beiden Funktionen können Sie eine Codierungstabelle übergeben, die festlegt, welche Zeichen auf welche Entities abgebildet werden sollen. Die in den obigen Funktionen verwendeten Tabellen erhalten Sie mit der Funktion `get_html_translation_table()`, der Sie `HTML_ENTITIES` oder `HTML_SPECIALCHARS` übergeben. Es wird dann ein Array zurückgegeben, das Zeichen auf Entities abbildet; dieses können Sie dann als Grundlage für eine eigene Tabelle verwenden.

```
$copyright = "Copyright © 2003 O'Reilly & Associates\n";  
$table = get_html_translation_table(); // enthält <, >, " und &  
$table[0] = '&copy;'; // fügt © hinzu  
print strtr($copyright, $table);  
Copyright &copy; 2003 O'Reilly & Associates
```

Siehe auch

Rezepte 16.8, 17.12 und 17.16; die Dokumentationen zu `htmlentities()` unter <http://www.php.net/htmlentities> und `htmlspecialchars()` unter <http://www.php.net/htmlspecialchars>.

11.9 Mit Formularvariablen arbeiten, deren Name einen Punkt enthält

Problem

Sie möchten eine Variable verarbeiten, in deren Name sich ein Punkt befindet. Wenn das Formular abgesendet wird, können Sie aber die Variable nicht finden.

Lösung

Ersetzen Sie den Punkt im Variablennamen durch einen Unterstrich. Wenn Sie beispielsweise ein Formular-Eingabeelement namens `foo.bar` haben, greifen Sie darauf innerhalb von PHP unter dem Variablennamen `$_REQUEST['foo_bar']` zu.

Diskussion

Da PHP den Punkt als Operator zur String-Verkettung verwendet, wird eine Variable mit dem Namen `tier.groesse` automatisch in `tier_groesse` konvertiert; dadurch wird eine Mehrdeutigkeit für den Parser vermieden. Bei `$_REQUEST['tier.groesse']` kann diese Mehrdeutigkeit zwar nicht mehr entstehen, aber aus Kompatibilitäts- und Konsistenzgründen geschieht dies unabhängig davon, wie Sie `register_globals` eingestellt haben.

Normalerweise haben Sie mit dieser automatischen Namenskonvertierung zu tun, wenn Sie ein Bild zum Absenden eines Formulars verwenden. Ein Beispiel: Sie haben eine Straßenkarte mit den Standorten Ihrer Geschäfte, und Sie möchten, dass die Anwender auf einen der Standorte klicken, um weitere Informationen zu erhalten. Hier ist ein Beispiel:

```
<input type="image" name="locations" src="locations.gif">
```

Wenn ein Benutzer auf das Bild klickt, werden die x- und y-Koordinaten in der Form von `locations.x` und `locations.y` übertragen. Um herauszufinden, wo ein Benutzer geklickt hat, müssen Sie daher in PHP die Variablen `$_REQUEST['locations_x']` und `$_REQUEST['locations_y']` prüfen.

Mithilfe einer Reihe von Manipulationen kann man auch in PHP eine Variable mit einem Punkt im Namen erzeugen:

```
`${"a.b"} = 123; // durch {} erzwungen  
  
$var = "c.d"; // indirekte Benennung einer Variablen
```

```
$$var = 456;

print ${"a.b"} . "\n";
print $$var . "\n";
123
456
```

Aufgrund der umständlichen Syntax ist so etwas jedoch eher verpönt.

Siehe auch

Die Dokumentation zu externen PHP-Variablen unter <http://www.php.net/manual/language.variables.external.php>.

11.10 Formularelemente mit Mehrfachoptionen verwenden

Problem

Sie haben ein Formularelement mit mehreren Werten, zum Beispiel ein checkbox- oder select-Element, PHP sieht aber nur einen Wert.

Lösung

Schreiben Sie eckige Klammern ([]) hinter den Variablennamen:

```
<input type="checkbox" name="boroughs[]" value="bronx"> die Bronx
<input type="checkbox" name="boroughs[]" value="brooklyn"> Brooklyn
<input type="checkbox" name="boroughs[]" value="manhattan"> Manhattan
<input type="checkbox" name="boroughs[]" value="queens"> Queens
<input type="checkbox" name="boroughs[]" value="statenisland"> Staten Island
```

Innerhalb Ihres Programms behandeln Sie die Variable als Array:

```
print 'Ich mag ' . join(' und ', $boroughs) . '!';
```

Diskussion

Wenn Sie [] hinter einem Variablennamen einfügen, behandelt PHP die Variable als Array und nicht als Skalar. Wird einer solchen Variablen ein weiterer Wert zugewiesen, vergrößert PHP automatisch die Größe des Arrays und fügt den neuen Wert am Ende hinzu. Wenn also die ersten drei Felder in der Lösung angekreuzt sind, hat das die gleiche Wirkung, als hätten Sie den folgenden Code am Anfang des Skripts geschrieben:

```
$boroughs[] = "bronx";
$boroughs[] = "brooklyn";
$boroughs[] = "manhattan";
```

Auf diese Weise können Sie auch Informationen aus einer Datenbank zurückgeben, wenn mehrere Datensätze gefunden werden:

```
foreach ($_GET['boroughs'] as $b) {
    $boroughs[] = strtr($dbh->quote($b),array('_' => '\\_', '%' => '\\%'));
}
$locations = join(', ', $boroughs);

$dbh->query("SELECT address FROM locations WHERE borough IN ($locations)");
```

Diese Syntax funktioniert auch bei multidimensionalen Arrays:

```
<input type="checkbox" name="population[NY][NYC]" value="8008278">New York...
```

Wenn dieses Formularelement angekreuzt ist, wird der Wert von `$population['NY']['NYC']` auf 8008278 gesetzt.

Ein Variablenname mit angefügten `[]` kann zu Problemen führen, wenn Sie versuchen, das Element in JavaScript anzusprechen. Anstatt das Element mit seinem Namen zu adressieren, verwenden Sie hier die numerische ID. Sie können den Elementnamen auch in einfache Anführungszeichen platzieren. Eine weitere Möglichkeit besteht darin, dem Element eine ID zuzuweisen, etwa den Namen ohne die `[]`, und stattdessen diese ID zu verwenden. Bei dem folgenden Formular:

```
<form>
<input type="checkbox" name="myName[]" value="myValue" id="myName">
</form>
```

referenzieren die folgenden drei Ausdrücke dasselbe Formularelement:

```
document.forms[0].elements[0];           // numerische ID
document.forms[0].elements['myName[]'];  // Name in Anführungszeichen
document.forms[0].elements['myName'];    // zugewiesene ID
```

Siehe auch

Die Einführung zu Kapitel 4 mit weiteren Informationen zu Arrays.

11.11 Drop-down-Menüs auf Basis des aktuellen Datums erzeugen

Problem

Sie möchten eine Reihe von Drop-down-Menüs erstellen, die automatisch auf dem aktuellen Datum stehen.

Lösung

Finden Sie mit `date()` das aktuelle Datum in der Zeitzone des Webservers heraus und durchlaufen Sie die Tage mit `mktime()`.

Der folgende Code generiert option-Werte für den heutigen Tag sowie die sechs folgenden Tage. Im diesem Fall ist »heute« der 1. Januar 2002.

```
list($hour, $minute, $second, $month, $day, $year) =
    split(':', date('h:i:s:m:d:Y'));

// Eine Woche in einzelnen Tagen
for ($i = 0; $i < 7; ++$i) {
    $timestamp = mktime($hour, $minute, $second, $month, $day + $i, $year);
    $date = date("D, F j, Y", $timestamp);

    print "<option value=\"\$timestamp\">$date</option>\n";
}
<option value="946746000">Tue, January 1, 2002</option>
<option value="946832400">Wed, January 2, 2002</option>
<option value="946918800">Thu, January 3, 2002</option>
<option value="947005200">Fri, January 4, 2002</option>
<option value="947091600">Sat, January 5, 2002</option>
<option value="947178000">Sun, January 6, 2002</option>
<option value="947264400">Mon, January 7, 2002</option>
```

Diskussion

In der Lösung setzen wir den value für jedes Datum auf die Unix-Zeitstempel-Repräsentation, da wir damit innerhalb unseres Programms einfacher umgehen können. Natürlich können Sie aber ein anderes Format verwenden, das Ihnen am nützlichsten und am besten geeignet erscheint.

Geben Sie nicht der Versuchung nach, die mktime()-Aufrufe wegzulassen. Datums- und Zeitwerte sind nicht so konsistent, wie Sie vielleicht hoffen. Je nachdem, was Sie tun, erhalten Sie möglicherweise nicht das erwartete Ergebnis. Ein Beispiel:¹

```
$timestamp = mktime(0, 0, 0, 10, 24, 2002); // 24. Oktober 2002
$one_day = 60 * 60 * 24; // Anzahl der Sekunden eines Tages

// Die Woche als einzelne Tage ausgeben.
for ($i = 0; $i < 7; ++$i) {
    $date = date("D, F j, Y", $timestamp);

    print "<option value=\"\$timestamp\">$date</option>";

    $timestamp += $one_day;
}
<option value="972619200">Fri, October 25, 2002</option>
<option value="972705600">Sat, October 26, 2002</option>
<option value="972792000">Sun, October 27, 2002</option>
<option value="972878400">Sun, October 27, 2002</option>
```

¹ Die Funktion date() formatiert das Datum in englischer Sprache. Mehr über die Lokalisierung von Datumsangaben finden Sie in Kapitel 19.

```
<option value="972964800">Mon, October 28, 2002</option>
<option value="973051200">Tue, October 29, 2002</option>
<option value="973137600">Wed, October 30, 2002</option>
```

Dieses Skript soll eigentlich den Tag, den Monat und das Jahr für eine Periode von sieben Tagen ausgeben, die am 24. Oktober 2002 beginnt. Sie funktioniert aber nicht wie erwartet.

Warum gibt es zwei »Sun, October 27, 2002«? Die Antwort liegt in der Sommerzeit. Es trifft nicht zu, dass die Anzahl der Sekunden eines Tages konstant bleibt; tatsächlich ändert sich diese Zahl fast mit Sicherheit. Besonders problematisch ist dabei, dass Sie, wenn Sie nicht gerade in der Nähe eines der Übergangstage sind, den Fehler beim Testen kaum finden werden.

Siehe auch

Kapitel 3, zum Teil Rezept 3.14, aber auch die Rezepte 3.1, 3.2, 3.4, 3.11 und 3.15; die Dokumentationen zu `date()` unter <http://www.php.net/date> und `mktime()` unter <http://www.php.net/mktime>.