

Leseprobe

Erfahren Sie alles über PHP und moderne Datenbankentwicklung. In dieser Leseprobe lernen Sie die Programmiergrundlagen von PHP und den wichtigen Einsatz von Strings kennen. Außerdem erhalten Sie das vollständige Inhalts- und Stichwortverzeichnis des Buches.

»Grundlagen der Sprache«
»Programmieren«
»Strings«

Inhalt

Index

Die Autoren

Leseprobe weiterempfehlen

Christian Wenz, Tobias Hauser

PHP 5.6 und MySQL – Das umfassende Handbuch

1.047 Seiten, gebunden, März 2015
39,90 Euro, ISBN 978-3-8362-3058-2

www.rheinwerk-verlag.de/3714

Kapitel 4
Grundlagen der Sprache

Dieses Kapitel beginnt mit der Syntax von PHP und zeigt Ihnen, wie Sie mit PHP programmieren. Für fortgeschrittene Leser bildet es ein gutes Nachschlagewerk, wenn einzelne Konstrukte unklar sind.

PHP ist nicht schwer zu erlernen. Dieses Versprechen steht am Anfang einer umfangreichen Spracheinführung, die alle wesentlichen Aspekte der Sprache beleuchten wird. Sie finden viele kleine, einfache Codestücke. Dadurch wird es möglich, dass Sie auch später schnell einzelne Fakten nachschlagen und so immer tiefer in PHP einsteigen.

Tipp

Sollten Sie es eilig haben und seltenere Details nicht benötigen, sondern schnell und kompakt die Sprache lernen wollen, lassen Sie beim ersten Lesen einfach Überschriften der vierten Ebene weg. Dort finden Sie meist Hintergrundinformationen zu einzelnen Themen, die aber erst im Einzelfall wirklich wichtig werden.

4.1 PHP in HTML

Eine Reise in dieTiefen und Untiefen von PHP beginnt bei HTML. PHP wurde als serverseitige Programmiersprache konzipiert, die eng in HTML integriert ist. Dies steht im Gegensatz zum Ziel anderer Programmiersprachen, Code und Inhalt zu trennen. Natürlich ist eine solche Trennung auch in PHP möglich, indem Sie Code in eine externe PHP-Datei einschließen.[1] Häufiger aber wird der PHP-Code direkt in die HTML-Datei eingefügt. Die Datei erhält dabei die Endung *.php*, *.php4* oder *.php5*. Gebräuchlich ist heute hauptsächlich der Einsatz von *.php*.[2]

PHP-Anweisungen können in diese Dateien auf verschiedene Arten eingebunden werden:

```
<?php
  //Code
?>
```

1 Siehe Abschnitt »Externe Datei«.

2 Ab und an findet sich auch noch *.php3* für Dateien, die mit PHP 3 geschrieben wurden.

- Dies ist die Standardvariante, PHP-Code einzubinden. Auch Großschreibung ist erlaubt: `<?PHP`. Außerdem wird in vielen modernen Projekten (u. a. auch bei Frameworks wie Zend Framework und Symphony) das schließende Element `?>` in Dateien nur mit PHP-Code, also ohne HTML, weggelassen, da dahinter aus Versehen noch Whitespace folgen kann, der dann zu einer Fehlermeldung in Form eines `Cannot modify header`-Fehlers führt.

  ```
  <?  //Code  ?>
  ```

  Ein wenig kürzer geht es, wenn Sie `php` einfach weglassen und nur spitze Klammern und Fragezeichen verwenden. Allerdings ist diese Variante nicht XML-konform und kann in der *php.ini* über die Option `short_open_tag = Off` ausgeschaltet werden. Standardmäßig steht hier zwar `On`, aber dennoch sollten Sie sich nicht darauf verlassen.

  ```
  <%
    //Code
  %>
  ```

- Diese Form entspricht ASP (*A*ctive *S*erver *P*ages), der – inzwischen veralteten[3] – serverseitigen Programmiertechnologie von Microsoft. Damit diese Variante in PHP zur Verfügung steht, müssen Sie den Eintrag `asp_tags` in der Konfigurationsdatei *php.ini*[4] auf `On` setzen. Über eine Abschaffung wird bereits seit längerer Zeit diskutiert, Sie sollten diese Form dementsprechend nicht mehr einsetzen.

  ```
  <script language="php">
    //Code
  </script>
  ```

- Die letzte Form ist in der Praxis ungebräuchlich, da sie sehr viel Tipparbeit bedeutet. Sie funktioniert allerdings immer und entspricht der Einbindung eines clientseitigen Skripts mit JavaScript.

Allen Arten gemeinsam ist, dass es sich um PHP-Anweisungsblöcke handelt. Sie können beliebig viele PHP-Blöcke in eine HTML-Seite einbauen.

Hinweis

Wenn in einer PHP-Seite keine PHP-Anweisungen gefunden werden, gibt der PHP-Interpreter einfach den HTML-Code aus.

3 Abgelöst durch ASP.NET. Dort wird Code anders eingebunden.
4 Mehr zur Konfiguration von PHP finden Sie in Kapitel 37, »Apache-Funktionen«.

4.1.1 Kommentare

Ein Kommentar ist Text im Quellcode, der vom PHP-Interpreter nicht ausgeführt wird. Kommentare dienen in der Praxis dazu, Teile des Codes vernünftig zu erklären oder sonstige Informationen mitzuliefern. PHP verwendet eine Syntax für Kommentare, die Sie vielleicht schon aus JavaScript oder anderen Sprachen kennen:

```
// Kommentar
```

steht für einen einzeiligen Kommentar. Alle Zeichen nach // sind auskommentiert.

```
# Kommentar
```

steht ebenfalls für einen einzeiligen Kommentar.

```
/* Mehrzeiliger
Kommentar */
```

kommentiert einen Block zwischen /* und */ aus, der sich auch über mehrere Zeilen erstrecken darf.

Tipp

Kommentieren Sie Ihren Code sinnvoll und verständlich. Denken Sie einfach an den armen Kollegen, der daran weiterarbeiten muss, oder an sich selbst, wie Sie nach Jahren vergessen haben, worum es sich bei dem Skript handeln sollte. In beiden Fällen werden Sie einen Menschen mit guten Kommentaren glücklich machen!

4.1.2 Anweisungen

Alle Zeichen innerhalb eines PHP-Anweisungsblocks, die nicht auskommentiert sind, bilden zusammen den PHP-Code, den der PHP-Interpreter ausführt. Jede Zeile in PHP, die eine Anweisung enthält, wird mit einem Strichpunkt beendet:

```
<?php
  echo "Text";
?>
```

gibt beispielsweise einen Text aus.

Hinweis

Zur Anweisung gehört auch der Begriff *Ausdruck* (engl.: *Expression*). In PHP ist alles ein Ausdruck, was einen Wert besitzt. Die meisten Anweisungen sind insofern auch Ausdrücke. Diese Definition ist allerdings eher akademisch und für Ihre praktische Arbeit wohl nur selten relevant.

4.1.3 Externe Datei

Die Trennungvon Code und Inhalt gehört zwar nicht zu den ursprünglichen Intentionen von PHP, ist allerdings über externe Dateien zu realisieren.[5] Auch sonst sind externe Dateien praktisch. Sie erlauben auch, häufig verwendete Codestücke auszulagern.

Zum Einbinden von externen Dateien verwenden Sie die Anweisungen `include()` und `require()`. Funktional unterscheiden sich beide beim Fehlerhandling. `include()` produziert nur eine Warnung (`E_Warning`), wenn beispielsweise die externe Datei nicht gefunden wird, `require()` liefert einen Fehler (`E_Error`). Dies ist vor allem beim Fehler-Handling und bei den Konfigurationseinstellungen für Fehlertoleranz in der *php.ini* wichtig.[6]

Ein einfaches Beispiel illustriert die Funktionsweise der beiden Anweisungen. Die externe Datei enthält eine Ausgabe mit der `echo`-Anweisung:

```
<?php
   echo "Externe PHP-Datei!";
?>
```

Listing 4.1 Die externe Datei gibt einen Text aus (»extern.php«).

Tipp

PHP-Code muss ganz normal in einen PHP-Anweisungsblock eingeschlossen werden. Zusätzlich kann die externe Datei HTML-Quellcode enthalten. Wenn der PHP-Interpreter eine externe Datei aufruft, liest er das HTML ein und interpretiert die PHP-Blöcke.

5 Beim Vergleich serverseitiger Technologien ist die Trennung von Code und Inhalt – eine Form des modularen Programmierens – eine wichtige Forderung, die beispielsweise ASP.NET sehr gut erfüllt. Allerdings muss man bedenken, dass PHP ursprünglich gerade in der engen Integration von PHP-Code und HTML-Code einen Vorteil gegenüber dem damaligen Marktführer Perl hatte. Dank externer Dateien können Sie mit PHP mittlerweile aber sowohl »getrennt« als auch »integriert« programmieren, sodass bei sauberer Programmierung kein Unterschied mehr besteht.

6 Mehr hierzu lesen Sie in Kapitel 35. Im Testbetrieb sollten Sie `error_reporting` in der *php.ini* immer auf `E_ALL` belassen, damit alle Fehlermeldungen angezeigt werden und Sie Probleme schnell erkennen können. Außerdem sollten Sie in der Entwicklungs- und Testumgebung die Fehlermeldungen mit `display_errors =On` einblenden.

Diese Datei wird dann mit `include()` in eine Datei eingebaut:

```
<html>
  <head>
    <title>PHP-Einbau</title>
  </head>
  <body>

    <?php
      include "extern.php";
    ?>

  </body>
</html>
```

Listing 4.2 »include()« bindet die externe Datei ein (»include.php«).

Befindet sich die Datei nicht im selben Verzeichnis oder in einem Verzeichnis, das per `include_path`-Direktive in der *php.ini* angegeben ist, müssen Sie den vollständigen Pfad zur Datei angeben.

Hinweis

Windows unterscheidet bei Dateinamen nicht zwischen Groß- und Kleinschreibung. Insofern unterscheiden auch die Befehle zum Einbinden externer Dateien unter Windows z. B. nicht zwischen *extern.php* und *Extern.php*.

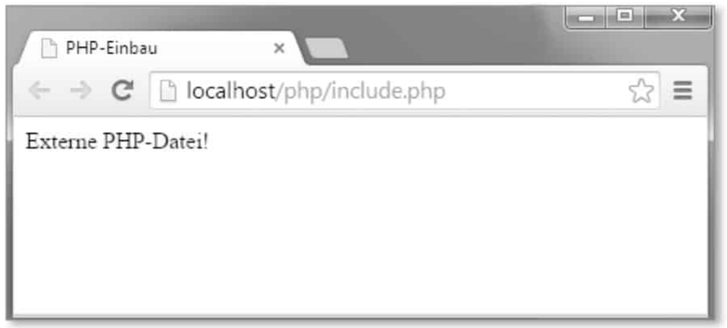


Abbildung 4.1 Der Inhalt der externen Datei wird ausgegeben.

Die Syntax mit `require()` sieht genauso aus:

```
require "extern.php";
```

Hinweis

Anweisungen[7] sind von PHP angebotene Sprachkonstrukte, um ein bestimmtes Ziel zu erreichen. Die Parameter für Anweisungen werden in Anführungszeichen nach der Anweisung geschrieben. Alternativ ist hier auch eine Syntax mit runden Klammern möglich:

```
require("extern.php");
```

»include_once« und »require_once«

Neben `include()` und `require()` gibt es noch `include_once()` und `require_once()`. Diese beiden Sprachkonstrukte prüfen zuerst, ob die Datei bereits eingefügt wurde. Sollte sie schon eingebunden worden sein, geschieht dies nicht noch einmal.

Dieses Verhalten ist dann wünschenswert, wenn Ihr Skript wirklich Gefahr läuft, eine Datei mehrmals einzulesen. In diesem Fall kann es nämlich sein, dass bestehende Variablenwerte oder Funktionen erneut überschrieben werden bzw. bei Funktionen ein Fehler erscheint, da sie im selben Kontext immer nur einmal deklariert werden können.

Der Einsatz von `include_once()` und `require_once()` erfolgt genau wie der von `include()` und `require()`:

```
include_once "extern.php";
```

bzw.:

```
require_once "extern.php";
```

Rückgabewert

Liefert das Skript in der externen Datei einen Rückgabewert mit `return`[8], kann dieser auch in einer Variablen[9] gespeichert werden:

```
$wert = require("extern.php");
```

7 Hier ist die Nomenklatur nicht eindeutig. Eine Zeile in PHP, die mit einem Strichpunkt endet, heißt ebenfalls Anweisung. Sie enthält sogar meist ein PHP-Sprachkonstrukt, also eine Anweisung im engeren Sinne (alternativ: Befehl). Die Unterscheidung der Begriffe ist allerdings eher akademischer Natur und hat auf die Praxis keine Auswirkungen.

8 Siehe Kapitel 6, »Funktionen und Sprachkonstrukte«.

9 Siehe Abschnitt 4.3, »Variablen«.

Besonderheiten in »if«-Anweisungen und Schleifen

Wird eine `include()`- oder `require()`-Anweisung in anderen Anweisungen wie `if`-Bedingungen oder Schleifen[10] eingebettet, *muss* diese Anweisung geschweifte Klammern besitzen, also ein abgeschlossener Block sein. Die Kurzform:

```
if (Bedingung)
  include "extern.php";
else
  include "extern2.php";
```

ist also nicht erlaubt, funktioniert allerdings in manchen PHP-Versionen dennoch. Korrekt ist:

```
if (Bedingung) {
  include "extern.php";
}
else {
  include "extern2.php";
}
```

Dateien über das Netzwerk

Wenn Sie Dateien über das Netzwerk mit absoluter URL öffnen möchten, muss in der *php.ini*-Konfigurationsdatei die Einstellung `allow_url_fopen` aktiviert sein.[11]

```
allow_url_fopen = On
```

Hinweis

In der Windows-Version von PHP funktioniert dies erst ab Versionsnummer 4.3.0, nicht aber mit den vorherigen Versionen!

»include_path«

In der *php.ini* findet sich noch eine zweite interessante Einstellung: Unter `include_path` legen Sie beliebige Pfade fest, in denen `include()`- und `require()`-Anweisungen automatisch nachsehen. Mehrere Pfade werden unter Linux mit Doppelpunkt, unter Windows mit Strichpunkt getrennt. Hier die Linux-Variante:

```
include_path = ".:/php/includes"
```

10 Mehr Details dazu in Kapitel 5, »Programmieren«.

11 Siehe hierzu auch das Kapitel 33, »Sicherheit«.

Und hier die Windows-Variante:

```
include_path = ".;c:\php\includes"
```

Die Konstante `PATH_SEPARATOR` enthält das Trennzeichen je nach Betriebssystem. Damit müssen Sie sich also nicht um dieses Detail kümmern, sondern schreiben einfach:

```
include_path = "." . PATH_SEPARATOR . "c:\php\includes"
```

Sie können die Einstellung `include_path` auch für das aktuelle Skript ändern. Dazu gibt es zwei verschiedene Wege:

- die Funktion `set_include_path()`, die allerdings erst ab PHP-Version 4.3.0 vorhanden ist:

  ```
  set_include_path("/includes");
  ```

- die Funktion `ini_set()`, um jede beliebige Einstellung der *php.ini* zu ändern. Sie gibt es auch schon vor 4.3.0.

  ```
  ini_set("include_path", "/includes");
  ```

4.2 Ausgabe mit PHP

Um richtig in PHP einzusteigen, müssen Sie testen können, wie die Syntax und die Programmierkonstrukte funktionieren. Dazu sollten Sie Daten ausgeben können. PHP besitzt zwei Sprachkonstrukte[12] für die Ausgabe:

- die `echo`-Anweisung:

  ```
  <?php
    echo "Ausgabe";
  ?>
  ```

- die `print`-Anweisung:

  ```
  <?php
    print "Ausgabe";
  ?>
  ```

Die beiden Anweisungen unterscheiden sich dadurch, dass `echo` einfach nur das Übergebene ausgibt, `print` dagegen einen Rückgabewert liefert.[13]

12 Ein *Sprachkonstrukt* (engl.: *Statement*) ist eine Anweisung von PHP. Dieses Buch trennt zwischen Sprachkonstrukten (synonym: Anweisungen, Sprachanweisungen) und Funktionen. Mehr hierzu in Kapitel 6, »Funktionen und Sprachkonstrukte«.

13 Dieser Unterschied rührt davon her, dass `print` eigentlich ein Operator ist. Lesen Sie hierzu auch den Abschnitt »print« im nächsten Kapitel.

Dieser Rückgabewert kann in eine Variable (siehe nächster Abschnitt 4.3, »Variablen«) gespeichert werden. Er beträgt `1`, wenn die Ausgabe funktioniert hat, und `0`, wenn sie misslungen ist.

```
<?php
  $t = print "Ausgabe";
  echo $t;
?>
```

Listing 4.3 Rückgabewert von »print« (»print.php«)

Dieses Listing gibt

```
Ausgabe1
```

aus. In der Praxis kommt der Rückgabewert recht selten zum Einsatz.

> **Hinweis**
>
> PHP hält noch weitere Sprachanweisungen zur Ausgabe für Sie bereit. Sie finden sie in Kapitel 7, »Strings«, im Abschnitt »Ausgabe«.

Kurzfassung

Noch kürzer geht es, wenn Sie nur ein Gleichheitszeichen direkt nach dem Beginn des PHP-Blocks angeben:

```
<?="Kurze Ausgabe"?>
```

> **Tipp**
>
> Bis zur PHP-Version 5.3 musste `short_open_tags` auf `on` gesetzt sein, um die Kurzform zu verwenden. Seit Version 5.4 ist `<?=` auch verfügbar, wenn `short_open_tags` deaktiviert ist.

4.2.1 Anführungszeichen

Da die Ausgabe in Anführungszeichen erfolgt[14], ist die Frage, wie Anführungszeichen in der Zeichenkette behandelt werden. PHP erlaubt einfache und doppelte Anführungszeichen, um Ausgaben (respektive Zeichenketten) zu begrenzen.

14 Sie ist eine Zeichenkette (auch String). Mehr dazu im nächsten Abschnitt.

Sie können also

```
echo "Ausgabe";
```

oder

```
echo 'Ausgabe';
```

schreiben.

Um doppelte oder einfache Anführungszeichen zu verwenden, müssen Sie die jeweils andere Anführungszeichen-Art einsetzen, um die Ausgabe zu begrenzen:

```
echo 'Er sagte: "Ich denke, also bin ich!"';
```

Die zugehörige Ausgabe sehen Sie in Abbildung 4.2.



Abbildung 4.2 Anführungszeichen in der Ausgabe

Wenn Sie einfache und doppelte Anführungszeichen in einem String verwenden möchten, müssen Sie die jeweiligen Anführungszeichen per Backslash entwerten:

```
echo 'McDonald\'s-Esser: "Ich liebe nichts!"';
```

Mehr zum Entwerten lesen Sie im Abschnitt »Variablen ausgeben«.

4.3 Variablen

Eine Variable speichert einen Wert. Dieser Wert kann im Lauf eines Skripts geändert werden, er ist also variabel. Dieses Verhalten gibt der Variablen ihren Namen.

In PHP beginnen alle Variablen mit dem Dollarzeichen ($).[15] PHP erfordert – im Gegensatz zu anderen Programmiersprachen – nicht, dass eine Variable beim ersten

15 Diese Syntax lehnt sich an Perl (Practical Extraction and Report Language) an, der sehr mächtigen, aber teilweise auch recht komplizierten Skriptsprache. Insgesamt nimmt die Syntax von PHP viele Anleihen an Perl und übernimmt beispielsweise auch die regulären Ausdrücke.

Auftreten deklariert wird. Allerdings müssen Sie einer Variablen natürlich einen Wert zuweisen. Dies geht mit dem Gleichheitszeichen (=), dem sogenannten Zuweisungsoperator:

```
$text = "Wert";
```

weist also der Variablen `text` eine Zeichenkette mit dem Inhalt `"Wert"` zu.

4.3.1 Datentypen

Zeichenkettenwerden immer in Anführungszeichen geschrieben und heißen auch Strings. Zeichenketten sind allerdings nicht die einzigen Datentypen, die eine Variable annehmen kann. PHP unterscheidet außerdem noch folgende Datentypen:

- Integer (`integer` und `int`[16]) sind ganze Zahlen.

  ```
  $zahl = 5;
  ```

- Double ist der Datentyp für Fließkommazahlen. In Double sind auch die ganzen Zahlen enthalten.

  ```
  $kommazahl = 5.4;
  ```

Hinweis

Beachten Sie, dass in PHP Kommazahlen immer mit Dezimalpunkt statt dem deutschen Komma geschrieben werden. Da das Komma in der Sprachsyntax eine völlig andere Bedeutung hat, kommt es in der Praxis meist zu einer Fehlermeldung.

- Real ist eine andere Bezeichnung für Double.
- Boolean (`boolean` oder `bool`) steht für einen Wahrheitswert. Ein Boolean hat nur die Werte `true` (wahr) oder `false` (falsch). Wahrheitswerte sind beispielsweise die Ergebnisse von Bedingungen und Überprüfungen.

  ```
  $wahr = true;
  ```

- `Object` steht für ein Objekt in PHP. Nähere Informationen hierzu erfahren Sie in Kapitel 11, »Objektorientiert programmieren«.
- Arrays können mehrere Werte speichern und sind für die Programmierung sehr wichtig. Mehr zu Arrays lesen Sie in Kapitel 8, »Arrays«.
- `Resource` ist ein intern von PHP verwendeter Datentyp, in dem beispielsweise Zugriffe auf Datenquellen gespeichert werden.
- `NULL` steht für keinen Wert, ist aber selbst auch ein Datentyp.

16 Die Kurzformen `int` und `bool` gibt es seit Version 4.2.0 von PHP.

In den meisten Fällen müssen Sie sich nicht um den Datentyp kümmern, da PHP den Datentyp des Werts einer Variablen automatisch feststellt und ihn umwandelt, wenn er sich ändert. Die automatische Typkonvertierung funktioniert allerdings nicht immer wie erwartet und/oder gewünscht. Deswegen zeigen die nächsten beiden Unterabschnitte zuerst, wie Sie den Datentyp einer Variablen feststellen, und dann, wie Sie den Typ ändern.

Tipp

Sollten Sie schnell in PHP einsteigen wollen, überblättern Sie diese Abschnitte einfach und lesen Sie sie später nach, wenn Sie sie für Ihre Anwendung benötigen.

Datentyp feststellen

Mit der Funktion `gettype(Variable)` können Sie den Datentyp einer Variablen herausfinden. Sie erhalten als Rückgabe den Datentyp in langer Form, also z. B. `boolean` statt `bool`.

```
<?php
  $a = "Text";
  echo gettype($a);
?>
```

Listing 4.4 Den Datentyp feststellen (»datentyp.php«)

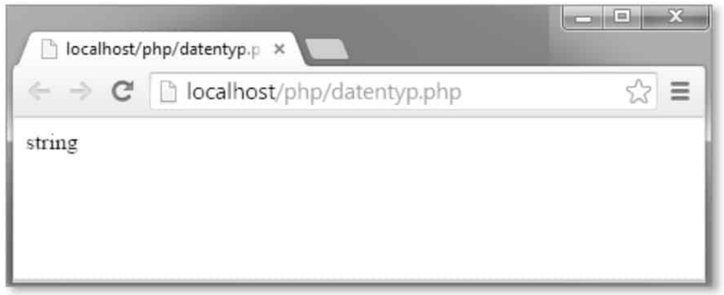


Abbildung 4.3 Die Variable hat den Datentyp »String«.

Hinweis

Neben der allgemeinen Funktion `gettype()` gibt es noch viele einzelne Funktionen, die auf jeweils einen bestimmten Datentyp testen. `is_bool()` prüft auf Boolean, `is_string()` auf String, `is_numeric()` darauf, ob es sich um eine Zahl handelt etc. Rückgabewert ist jeweils ein Wahrheitswert, `true`, wenn der Datentyp vorliegt, `false`, wenn nicht.

Typkonvertierung

Normalerweise müssen Sie sich in PHP um die Typkonvertierung nicht kümmern. Das folgende Skript würde in vielen Programmiersprachen die Zahl an den String anhängen. Da PHP allerdings für das Verbinden von Strings einen eigenen Operator, den Punkt (`.`), verwendet, funktioniert hier die Typkonvertierung richtig:

```
<?php
  $a = "3";
  $b = 5;
  $erg = $a + $b;
  echo $erg;
?>
```

Listing 4.5 Automatische Typkonvertierung (»typkonvertierung_auto.php«)

Das Ergebnis der Berechnung ist also:

```
8
```

Wenn Sie doch einmal Typkonvertierung benötigen, finden Sie in PHP die von C bekannte Typkonvertierung (engl.: *Type Casting*). Sie schreiben den Datentyp (in Kurz- oder Langform) vor die Variable, die umgewandelt werden soll.

```
<?php
  $a = "true";
  $b = (bool) $a;
  echo $b;
?>
```

Listing 4.6 Typkonvertierung mit PHP (»typkonvertierung.php«)

Als Ausgabe des obigen Skripts erfolgt der Wahrheitswert `1`, der für `true` steht:

```
1
```

Alternativ zur Konvertierung mit dem Datentyp vor der Variablen können Sie auch die Funktion `settype(Variable, Datentyp)` einsetzen. Der Datentyp wird dabei als String übergeben:

```
<?php
  $a = "true";
  $b = settype($a, "boolean");
  echo $b;
?>
```

Listing 4.7 »settype()« (»settype.php«)

Als Ausgabe erfolgt wie bei der Konvertierung oben die `1`.

4.3.2 Benennung

Der Name einer Variablen darf in PHP nur aus Buchstaben, Ziffern und Unterstrichen (_) bestehen. Beginnen darf er nur mit Buchstaben oder einem Unterstrich, *nicht* aber mit einer Ziffer.

Trotz dieser Einschränkungen gehört PHP bei Variablennamen zu den liberalsten Programmiersprachen: Die Namen von Sprachkonstrukten und Anweisungen wie `echo` oder `if` können als Variablennamen verwendet werden.[17]

```
$echo = "Wert";
echo $echo;
```

Obiger Code gibt `Wert` aus.

Dass etwas möglich ist, heißt natürlich nicht, dass man es auch verwenden sollte. Und so lässt der gute Programmierer von solchen »Experimenten« lieber die Finger. Auch sollten Sie Variablen immer aussagekräftig benennen. Eine Variable muss nicht nur aus drei Zeichen bestehen, und Durchnummerieren ist meist sehr unübersichtlich, vor allem wenn Sie ein Skript nachträglich erweitern.

Sie sollten die Namenskonventionen für Variablen in einem Projekt immer vorher festlegen. Hier einige Vorschläge:

- Bei zusammengesetzten Namen können Sie die einzelnen Worte mit einem Unterstrich (_) trennen:

  ```
  $wert_links = 5;
  ```

- oder das neue Wort mit einem großen Anfangsbuchstaben beginnen:[18]

  ```
  $wertLinks = 5;
  ```

- Alternativ lassen Sie jedes Wort mit einem großen Anfangsbuchstaben beginnen:[19]

  ```
  $WertLinks = 5;
  ```

4.3.3 Variable Variablen

Das Konzept der variablen Variablennamen funktioniert so: Sie weisen einer Variablen einen String zu. Diese Variable können Sie nun als Name für eine weitere Variable

17 Diese Namen heißen auch *Schlüsselwörter*. In den meisten Programmiersprachen lassen sich Schlüsselwörter nicht als Variablennamen verwenden.

18 Diese Variante wird nach der Programmiersprache Pascal auch *Pascal-Case* genannt. Pascal wiederum ist nach dem Mathematiker Blaise Pascal benannt.

19 Dieses Verfahren heißt auch *Camel Case*, benannt nach den Höckern eines Kamels, oder, in PHP gebräuchlicher, *studlyCaps*.

festlegen. Dadurch wird eine Variable erzeugt, die als Namen den String der ersten Variablen und als Wert den Wert der zweiten Variablen besitzt:

```
<?php
  $a = "text";
  $$a = "Text für die Ausgabe";
  echo $text;
?>
```

Listing 4.8 Der Variablenname als Variable (»variable_variablen.php«)

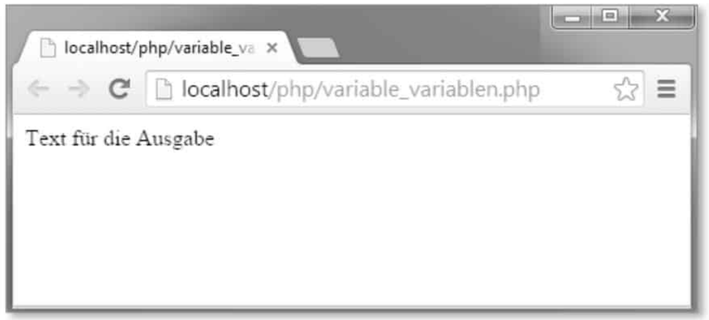


Abbildung 4.4 Der Text wird korrekt ausgegeben.

Hinweis

Das Zusammensetzen von Variablennamen macht vor allem dann Sinn, wenn Sie den Variablennamen dynamisch erzeugen möchten.

4.3.4 Variablen ausgeben

In den bisher gezeigten Beispielen wird oft der Wert einer Variablen mit `echo` (oder alternativ `print`) ausgegeben. Dies funktioniert problemlos:

```
$text = "Hallo PHP 5";
echo $text;
```

Die obigen Zeilen geben also Folgendes aus:

```
Hallo PHP 5
```

Sie können eine Variable allerdings auch in einer Zeichenkette ausgeben:

```
$text = "Hallo";
echo "$text PHP 5";
```

Diese zwei Zeilen produzieren als Ausgabe ebenfalls:

```
Hallo PHP 5
```

Dies funktioniert allerdings nur, wenn Sie doppelte Anführungszeichen verwenden. Bei einfachen Anführungszeichen wird dagegen die Variable nicht eingebunden:

```
$a = "Hallo";
echo '$a PHP 5';
```



Abbildung 4.5 Der Variablenname wird ausgegeben, da einfache Anführungszeichen zum Einsatz kommen.

Diese Unterscheidung zwischen doppelten und einfachen Anführungszeichen ist nicht nur beim Einsatz von Variablen relevant, sondern auch bei Escape-Sequenzen. Bei einer Escape-Sequenz wird ein Zeichen mittels des Backslashs (\) entwertet bzw. die Escape-Sequenz erzeugt eine bestimmte Wirkung. Die Unterscheidung zwischen doppelten und einfachen Anführungszeichen ist bei Escape-Sequenzen sehr einfach:

- Bei einfachen Anführungszeichen können Sie nur einfache Anführungszeichen und bei Bedarf den Backslash entwerten, wie Sie bereits gesehen haben.

  ```
  echo 'McDonalds\'-Esser: "Ich liebe nichts!"';
  ```

  Wenn Sie eine andere Escape-Sequenz einsetzen, wird diese nicht ausgeführt, sondern inklusive Backslash ausgegeben.

- Bei doppelten Anführungszeichen können Sie einfache Anführungszeichen sowieso verwenden, doppelte entwerten und zusätzlich einige Escape-Sequenzen einsetzen.

  ```
  $version = "PHP 5";
  echo "Die Variable \$version hat den Wert:\n $version";
  ```

  Wenn Sie die Ausgabe des Beispiels betrachten, sehen Sie, dass `\n` nicht ausgegeben wird. Dies liegt daran, dass `\n` nur einen Umbruch im Quellcode und nicht in HTML erzeugt. Wenn Sie im Webbrowser den Quellcode ansehen, erkennen Sie den Zeilenumbruch.


```
Die Variable $version hat den Wert:
PHP 5
```

Escape-Sequenz	Beschreibung
\\	Gibt einen Backslash aus. Selbiges erreichen Sie, wenn Sie nur einen Backslash ohne Escape-Stringfolge danach ausgeben.
\"	doppelte Anführungszeichen
\$	Dollarzeichen
\n	Zeilenumbruch (ASCII 10), allerdings nicht in HTML. Hierfür benötigen Sie das HTML-Tag ` `.
\r	Wagenrücklauf (ASCII 13)
\t	Tabulator (ASCII 9)
\000	Ein bis drei Ziffern stellen eine Zahl in oktaler Notation dar.* Das entsprechende Zeichen wird dann ausgegeben.
\x00	Ein x und ein oder zwei Ziffern bilden eine Zahl in hexadezimaler Notation.**

*) Oktale Notation: Basis des oktalen Systems ist die 8. Alle Ziffern gehen von 0 bis 7. Die Umrechnung erfolgt so: Aus 245 wird 2 * 64 + 4 * 8 + 5, und das ergibt 165.

**) Hexadezimale Notation: Das hexadezimale System schreibt Zahlen auf der Basis von 16. Deswegen gibt es 16 Ziffern, nämlich die von 0 bis 9 bzw. die Buchstaben A bis F. Eine hexadezimale Zahl aus zwei Ziffern rechnen Sie so um: Die erste Ziffer multiplizieren Sie mit 16 und addieren zum Ergebnis die zweite. Hexadezimale Zahlen kommen beispielsweise zur Farbnotation in HTML zum Einsatz.

Tabelle 4.1 Escape-Sequenzen für doppelte Anführungszeichen

4.3.5 Nützliches und Hilfreiches

In diesem Abschnitt sind Informationen versammelt, die Sie zum Arbeiten mit PHP nicht unbedingt brauchen, die aber für fortgeschrittene Aufgaben durchaus nützlich sind.

»isset()«

Die Hilfsfunktion `isset(Variable)` prüft, ob eine Variable existiert. Sie liefert als Ergebnis einen Wahrheitswert. Da es wenig spannend wäre, diesen Wahrheitswert einfach nur auszugeben, greifen wir ein wenig vor und zeigen bereits eine Fallunterscheidung, die erst im nächsten Kapitel genauer besprochen wird.

Das folgende Skript überprüft, ob eine Variable existiert. Wenn ja, wird sie ausgegeben. Ansonsten erscheint eine Alternativmeldung.

```
<?php
  $test = "Textvariable";
  if (isset($test)) {
    echo $test;
  } else {
    echo "Variable nicht gesetzt";
  }
?>
```

Listing 4.9 »isset()« (»isset.php«)

Im obigen Beispiel ist die Variable gesetzt und wird deswegen ausgegeben. Was aber, wenn Sie der Variablen gar keinen Wert zuweisen?

```
$test;
if (isset($test)) {
  echo $test;
} else {
  echo "Variable nicht gesetzt";
}
```

In diesem Fall wird der Alternativtext `Variable nicht gesetzt` ausgegeben.

> **Hinweis**
>
> `isset()` liefert auch `false`, wenn eine Variable den Wert `NULL` (kein Wert) hat.

»empty()«

Einen ähnlichen Test wie `isset()` führt `empty()` durch. `empty(Variable)` prüft, ob eine Variable leer ist. Eine leere Variable ist allerdings auch ein leerer String oder `null`. Hierin liegt der Unterschied zu `isset()`.

```
<?php
  $test = "";
  if (empty($test)) {
    echo "Variable ist leer";
  } else {
    echo $test;
  }
?>
```

Listing 4.10 »empty()« (»empty.php«)

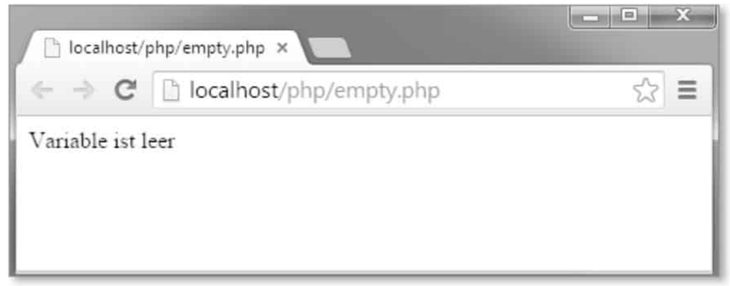


Abbildung 4.6 Hier liefert »empty()« »true«, da der String leer ist.

> **Hinweis**
>
> In der PHP-Dokumentation finden Sie eine recht interessante Vergleichstabelle der verschiedenen Testfunktionen (*http://www.php.net/manual/de/types.comparisons.php*).

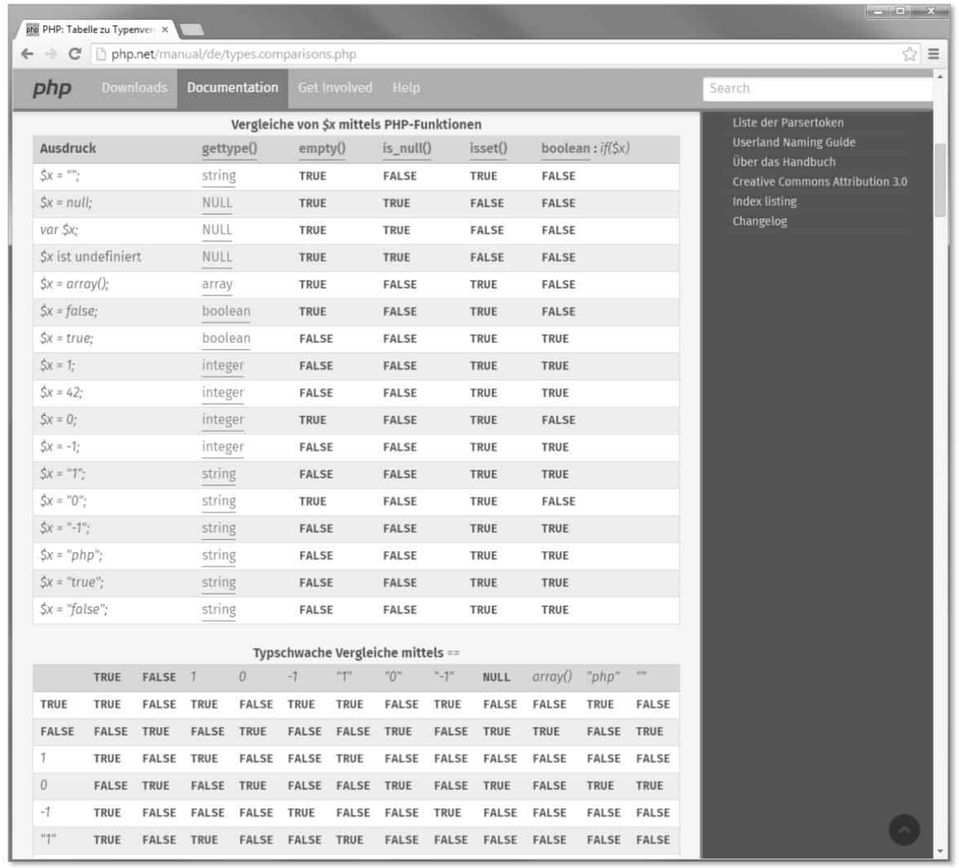


Abbildung 4.7 Der Vergleich der verschiedenen Funktionen ist sehr aufschlussreich, wenn Sie ein spezifisches Vergleichsproblem haben.

»is_null()«

Die Funktion `is_null(Variable)` gehört ebenfalls in die Riege der Hilfs- und Testfunktionen. Sie testet, ob eine Variable den Wert `NULL` (kein Wert) besitzt.

```
<?php
  $test = null;
  if (is_null($test)) {
    echo "Variable ist NULL";
  } else {
    echo "Variable ist nicht NULL, sondern" . $test;
  }
?>
```

Listing 4.11 »is_null()« (»is_null.php«)

Im obigen Fall ist die getestete Variable `null`, und deswegen wird Folgendes ausgegeben:

```
Variable ist NULL
```

Hinweis

Die Schreibweise der Funktionen in PHP ist leider teilweise etwas uneinheitlich. `isset()` wird zusammengeschrieben, `is_null()` hingegen mit Unterstrich. Dies hat historische Gründe: Die Funktionen wurden einfach irgendwann so genannt und konnten dann – um die Abwärtskompatibilität der Skripte zu erhalten – nicht mehr umbenannt werden.

In PHP 5 wurden die Funktionen und Methoden einiger APIs, z. B. zur XML-Steuerung, umbenannt. Die Kernfunktionen, zu denen auch die zur Variablenbehandlung gehören, sind in der alten Version geblieben.

»unset()«

Das Sprachkonstrukt `unset(Variable)` löscht eine Variable. Sie benötigen diese Funktion beispielsweise, wenn Sie bewusst im Hauptspeicher Platz schaffen möchten.

```
<?php
  $test = "Eine Variable.";
  echo $test;
  unset($test);
  echo $test;
?>
```

Listing 4.12 »unset()« (»unset.php«)

Dieses Beispiel gibt nur einmal den Text `Eine Variable.` aus. Bei der zweiten Ausgabe existiert die Variable schon nicht mehr. Hier zeigt PHP eine Notice `Undefined variable`.

Hinweis

Wenn Sie einen Parameter per Referenz an eine Funktion übergeben (siehe Kapitel 6, »Funktionen und Sprachkonstrukte«), wird mit `unset()` nur die lokale Variable, nicht aber das Original, auf das die Referenz verweist, gelöscht.

Referenzen

Normalerweise hat eine Variable genau einen Wert. Der Wert der Variablen wird vom PHP-Interpreter im Hauptspeicher gespeichert. Sie können allerdings auch mehrere Variablen auf einen Wert verweisen lassen. Das funktioniert mit dem Et-Zeichen, das oft auch kaufmännisches Und oder, auf Englisch, Ampersand genannt wird (&). Und so geht es: Sie erstellen eine Variable und weisen dann mithilfe des Et-Zeichens einer anderen Variablen die Referenz auf diese Variable zu:

```
<?php
  $a = "Eine Variable";
  $b = &$a;
  $a = "Geänderte Variable";
  echo $b;
?>
```

Listing 4.13 Referenz auf eine Variable (»variablen_referenz.php«)

Wenn Sie dann die ursprüngliche Variable, hier `$a`, ändern, erhält auch die Variable mit der Referenz, hier also `$b`, den neuen Wert. Übrigens, zwischen Ist-Gleich und Ampersand kann hier auch ein Leerzeichen folgen oder der Ampersand direkt vor der Variablen `$a` stehen.



Abbildung 4.8 Der geänderte Wert wird ausgegeben, da »$b« die Referenz darauf enthält.

4.3.6 Vordefinierte Variablen

Eine Sprache wie PHP besteht natürlich nicht nur aus dem Sprachkern. Um PHP herum gibt es eine große Umwelt – HTML-Formulare, Cookies, also kleine Textdateien im Browser, und vieles mehr. Für diese Umwelt, die Sie im Laufe dieses Buches noch kennenlernen werden, bietet PHP vordefinierte Variablen. Hier eine Auswahl:

- `$_GET` enthält die per GET aus einem Formular an die URL angehängten Werte.
- `$_POST` enthält die per POST von einem Formular versandten Werte.
- `$_COOKIE` enthält Informationen zu Cookies. Mehr dazu in Kapitel 13, »Entwurfsmuster: MVC & Co.«.
- `$_REQUEST` enthält die Informationen aus den oben genannten drei Variablen. Mehr dazu in Kapitel 13, »Entwurfsmuster: MVC & Co.«, und in Kapitel 14, »Formulare«.
- `$_SESSION` liefert Daten aus Session-Variablen. Mehr dazu in Kapitel 14.
- `$_SERVER` enthält Informationen über die PHP-Installation und den Webserver.
- `$_ENV` bietet Informationen über die Umgebung, in der PHP läuft.
- `$_FILES` besteht aus Daten über hochgeladene Dateien. Dazu finden Sie Informationen in Kapitel 24, »PostgreSQL«.
- `$GLOBALS` enthält alle globalen Variablen. Mehr dazu in Abschnitt 6.1.2, »Gültigkeit von Variablen«.

Hinweis

Diese vordefinierten Variablen heißen auch superglobale Arrays, da sie überall in PHP zur Verfügung stehen. Sie gibt es seit der PHP-Version 4.1.0. Davor existierten diese Arrays zwar auch schon, sie hießen aber anders und begannen immer mit `$HTTP_`, also beispielsweise `$HTTP_GET_VARS`. Mehr zu den superglobalen Arrays erfahren Sie in den einzelnen Kapiteln. Die wichtigsten lernen Sie in Kapitel 14, »Formulare«, kennen.

4.4 Konstanten

Konstanten haben, im Gegensatz zu Variablen, immer den gleichen Wert, der anfangs einmal festgelegt wird. In PHP definieren Sie Konstanten mit der Funktion `define()`. Eine Zuweisung von Konstanten, wie Sie sie vielleicht aus anderen Programmiersprachen kennen, ist nicht möglich.

```
define("Konstante", "Wert");
```


Der Zugriff auf die Konstante erfolgt jederzeit mit ihrem Namen:

```
echo Konstante;
```

gibt ihren Wert, hier also den String `Wert`, aus.

Alternativ greifen Sie auf Konstanten mit der Funktion `constant(Name)` zu:

```
echo constant("Konstante");
```

Diese Funktion kommt zum Einsatz, wenn der Konstantenname nur als Referenz beispielsweise in einer Variablen oder als Parameter einer Funktion gespeichert übergeben wird.

```
$Name = "Konstante";
constant($Name);
```

Hinweis

Beachten Sie, dass Konstanten im Gegensatz zu Variablen kein `$`-Zeichen besitzen. Außerdem gelten Konstanten automatisch im ganzen Skript.

Kapitel 5
Programmieren

Das Klischee vom ungewaschenen, langhaarigen Programmierer ist natürlich eine Mär ... meistens. Dennoch gehört zum Programmieren Enthusiasmus, der zwar nicht den Gang ins Badezimmer und zum Friseur verhindert, aber dennoch etwas Zeit kostet.

In diesem Kapitel lernen Sie die Syntax von PHP, die wichtigsten Sprachkonstrukte und Grundlagen kennen.

5.1 Operatoren

Operatoren haben vor allem eine Aufgabe: Sie sollen Daten miteinander verbinden. Die Daten, die verbunden werden, heißen *Operanden*. Ein Operator arbeitet mit einem, zwei oder drei Operanden.[1] Der häufigste Fall sind zwei Operanden. Bei einem Operanden handelt es sich um eine Variable oder ein *Literal*. Hier mit Literal:

```
  1       +       2
Operand Operator Operand
```

oder mit Variablen:

```
  $a      +      $b
Operand Operator Operand
```

5.1.1 Arithmetische Operatoren

Addition, Subtraktion, Multiplikation, Division, das sind die *arithmetischen Operationen*, die Sie aus dem Matheunterricht kennen. Sie lassen sich in PHP ganz einfach einsetzen:

```
<?php
  $a = 7;
  $b = 3;
```

1 Ein Operator mit einem Operanden heißt auch unär, einer mit zweien binär und der mit dreien ternär.

```
  $erg = $a * $b;
  echo $erg;
?>
```

Listing 5.1 Ein Operator im Einsatz (»operatoren.php«)

Arithmetische Operatoren sind nur auf Zahlen anwendbar. Neben den Operatoren für die Grundrechenarten und dem Minuszeichen für negative Zahlen gibt es noch den Modulo, der mit dem Prozentzeichen (%) dargestellt wird. Der Modulo gibt den ganzzahligen Rest einer Division an.

```
$a = 7;
$b = 3;
$erg = $a % $b;
```

Nach den obigen Zeilen hat die Variable `$erg` den Wert `1`. Das rechnen Sie so aus: 7 geteilt durch 3 ist 2 und ein paar Zerquetschte (genauer, ein Drittel). Das ganzzahlige Ergebnis der Division ist also 2. 2 mal 3 ist 6. Der ganzzahlige Rest der Division ist folglich 7 minus 6, was 1 ergibt. Schneller kommen Sie zu diesem Ergebnis, wenn Sie die Nachkommastelle (hier ein Drittel ≈ 0,33333333333) wieder mit 3 multiplizieren.

Die folgende Tabelle gibt einen Überblick über die arithmetischen Operatoren:

Operator	Beispiel	Beschreibung
+	`$erg = 7 + 3; //10`	Addition zweier Zahlen
-	`$erg = 7 - 3; //4`	Subtraktion zweier Zahlen
*	`$erg = 7 * 3; //21`	Multiplikation zweier Zahlen
/	`$erg = 7 / 3; //2.33333333333`	Division zweier Zahlen
%	`$erg = 7 %3; //1`	Berechnet den ganzzahligen Rest einer Division.

Tabelle 5.1 Die arithmetischen Operatoren

Kurzformen

Wenn Sie den Wert einer Variablen ändern möchten, können Sie das so tun:

```
$erg = 7;
$erg = $erg + 3;
```

Der letzte Schritt ist allerdings etwas lang. Deswegen existiert eine Kurzform, die den arithmetischen Operator direkt mit dem Zuweisungsoperator verbindet:


```
$erg = 7;
$erg += 3;
```

Diese Kurzformen gibt es für alle arithmetischen Operatoren. Sie sind in der folgenden Tabelle zusammengefasst:

Operator	Beispiel (»$erg = 7«)	Beschreibung
+=	`$erg += 3; //10`	Addition zweier Zahlen
-=	`$erg -= 3; //4`	Subtraktion zweier Zahlen
*=	`$erg *= 3; //21`	Multiplikation zweier Zahlen
/=	`$erg /= 3; //2.33333333333`	Division zweier Zahlen
%=	`$erg %= 3; //1`	Berechnet den ganzzahligen Rest einer Division.

Tabelle 5.2 Die Kurzformen

Inkrement und Dekrement

Es geht noch kürzer: Mit dem Inkrement (++) erhöhen Sie einen Wert um 1, mit dem Dekrement (--) verkleinern Sie ihn um 1. In den folgenden Zeilen erhöhen Sie `$a` von `7` auf `8`.

```
$a = 7;
$a++;
```

> **Hinweis**
>
> Inkrement und Dekrement kommen hauptsächlich bei Schleifen zum Einsatz (siehe Abschnitt 5.3, »Schleifen«).

Für Inkrement und Dekrement ist entscheidend, ob sie vor oder nach der Variablen platziert sind. Vor der Variablen heißt, dass das Inkrement vor den anderen Anweisungen ausgeführt wird. Im folgenden Beispiel erhält die Variable `$erg` das Ergebnis `11`, da die Variable `$a` vor der nachfolgenden Addition mit `$b` um 1 auf 8 erhöht wird.

```
$a = 7;
$b = 3;
$erg = ++$a + $b;
```

Stünde das Inkrement hinter der Variablen `$a`, würde diese erst nach der Anweisung erhöht:

```
$erg = $a++ + $b;
```

In diesem Fall beträgt `$erg` nur `10`. `$a` ist allerdings auf `8` gestiegen.

Exponential-Operator

Der Exponential-Operator ** ist neu in PHP 5.6. Seine Aufgabe ist, wie der Name schon verrät, die Exponentialrechnung. Hier ein einfaches Beispiel:

```
$a = 2;
$n = 4;
$erg = $a ** $n;
```

Nach den obigen Zeilen hat die Variable `$erg` den Wert `16`. Der zweite Operand ist dabei immer der Exponent, d. h., diese Rechnung entspricht 2^4.

Wie gewohnt gibt es ihn auch in der Kurzform:

```
$erg = 2;
$erg **= 4;
```

Strings verbinden

In vielen Programmiersprachen dient das Plussymbol nicht nur zum Verbinden von Zahlen, sondern auch zum Verbinden von Strings. Dies ist in PHP nicht so. Stattdessen kommt der Punkt (`.`) zum Einsatz:

```
<?php
  $a = "Alles neu ";
  $b = "macht der Mai";

  $erg = $a . $b;
  echo $erg;
?>
```

Listing 5.2 Strings verknüpfen (»string_konkatenation.php«)

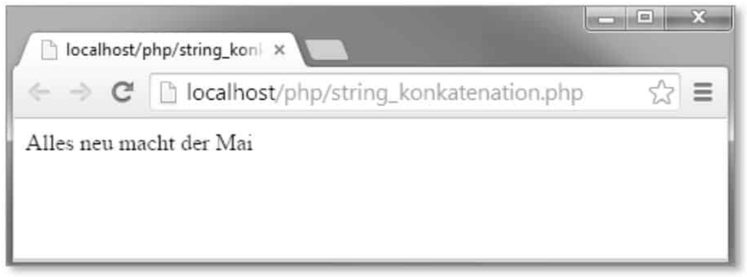


Abbildung 5.1 Die beiden Strings sind verbunden.

Hinweis

Das Verbinden von Strings heißt auch *Konkatenation*. Strings bieten noch viele andere Möglichkeiten. Mehr dazu lesen Sie in Kapitel 7, »Strings«.

Den Operator zum Verknüpfen von Strings gibt es auch in einer Kurzform in Verbindung mit dem Zuweisungsoperator:

```
$erg = "Alles neu ";
$erg .= "macht der Mai";
```

Die Variable `$erg` erhält als Wert `Alles neu macht der Mai`.

5.1.2 Vergleichsoperatoren

Wenn Sie mit PHP programmieren, werden Sie oft auf Fälle treffen, in denen Sie zwei Werte miteinander vergleichen müssen. Denken Sie beispielsweise an die Vollständigkeitsüberprüfung eines Formulars: dabei vergleichen Sie etwa, ob ein bestimmter Wert in ein Textfeld eingetragen wurde.

Für *Vergleiche* sind die Vergleichsoperatoren zuständig. Sie vergleichen zwei Operanden miteinander.

```
  7       >       3
Operand Operator Operand
```

Das Ergebnis ist ein Wahrheitswert (Boolean), also entweder `true` (wahr) oder `false` (falsch). Der obige Vergleich `7 > 3` ergibt also `true`, da 7 größer 3 ist. Wahrheitswerte werden von PHP bei der Ausgabe auch als Zahlen zurückgeliefert. `true` ist in diesem Fall `1`, `false` `0`.

Hinweis

Wenn Sie den Rückgabewert einer Operation mit Vergleichsoperator beispielsweise mit `echo` ausgeben, wird die `1` für `true` ausgegeben, die `0` für `false` aber nicht.

Die meisten Vergleichsoperatoren kennen Sie sicherlich schon. Die folgende Tabelle bietet einen Überblick:

Operator	Beispiel	Beschreibung
>	`$erg = 7 > 3 //true`	Größer als
<	`$erg = 7 < 3 //false`	Kleiner als

Tabelle 5.3 Die Vergleichsoperatoren

Operator	Beispiel	Beschreibung
>=	$erg = 3 >= 3 //true	Größer gleich
<=	$erg = 3 <= 3 //true	Kleiner gleich
==	$erg = 7 == 3 //false	Gleichheit
!=	$erg = 7 != 3 //true	Ungleichheit
<>	$erg = 7 <> 3 //true	Ungleichheit

Tabelle 5.3 Die Vergleichsoperatoren (Forts.)

Hinweis

Einer der häufigsten Fehler besteht darin, für die Prüfung der Gleichheit ein einfaches statt des doppelten Gleichheitszeichens zu verwenden. Dieser Fehler ist schwer zu entdecken, da beispielsweise in einer `if`-Anweisung (siehe Abschnitt 5.2, »Fallunterscheidungen«) ein einfaches Gleichheitszeichen als Zuweisung gewertet wird. Das heißt, PHP wirft also keine Fehlermeldung aus, sondern wertet den rechten Teil des Vergleichs als Wert der Variablen im linken Teil. Damit ist die Bedingung in der `if`-Anweisung außer bei `0` bzw. `false` immer erfüllt. Sie finden hierzu das Beispiel *gleichheit.php* in den Arbeitsdateien.

Genaue Gleichheit und Ungleichheit

Wenn Sie die Vergleichsoperatoren für Gleichheit und Ungleichheit um ein Gleichheitszeichen verlängern (aus == wird === und aus != wird !==), werden sie zur *genauen Gleichheit* und zur *genauen Ungleichheit*.[2] Dies bedeutet, beim Vergleich wird auch der Datentyp des Werts mit einbezogen.

```
$a = 3;
$b = "3";
$erg = $a === $b;
```

Welchen Wert hat die Variable `$erg`? Da die Variable `$a` eine Zahl ist und `$b` ein String, ist das Ergebnis `false`. Hätten Sie statt der genauen Gleichheit die einfache Gleichheit gewählt:

```
$erg = $a == $b;
```

wäre das Ergebnis `true`.

2 Die genaue Gleichheit und die genaue Ungleichheit heißen auch Identität und Nichtidentität.

Strings vergleichen

Wollen Sie zwei Strings miteinander vergleichen, so ist dies zwar möglich, allerdings mit einigen Problemen behaftet. Grundlage eines String-Vergleichs ist der ASCII-Code des jeweiligen Zeichens. ASCII steht für *American Standard Code for Information Interchange*. Dieser Code legt für die wichtigsten Zeichen und Buchstaben eine Zahl fest. Die Buchstaben beginnen ab Position 65 im ASCII-Code mit dem großen A.

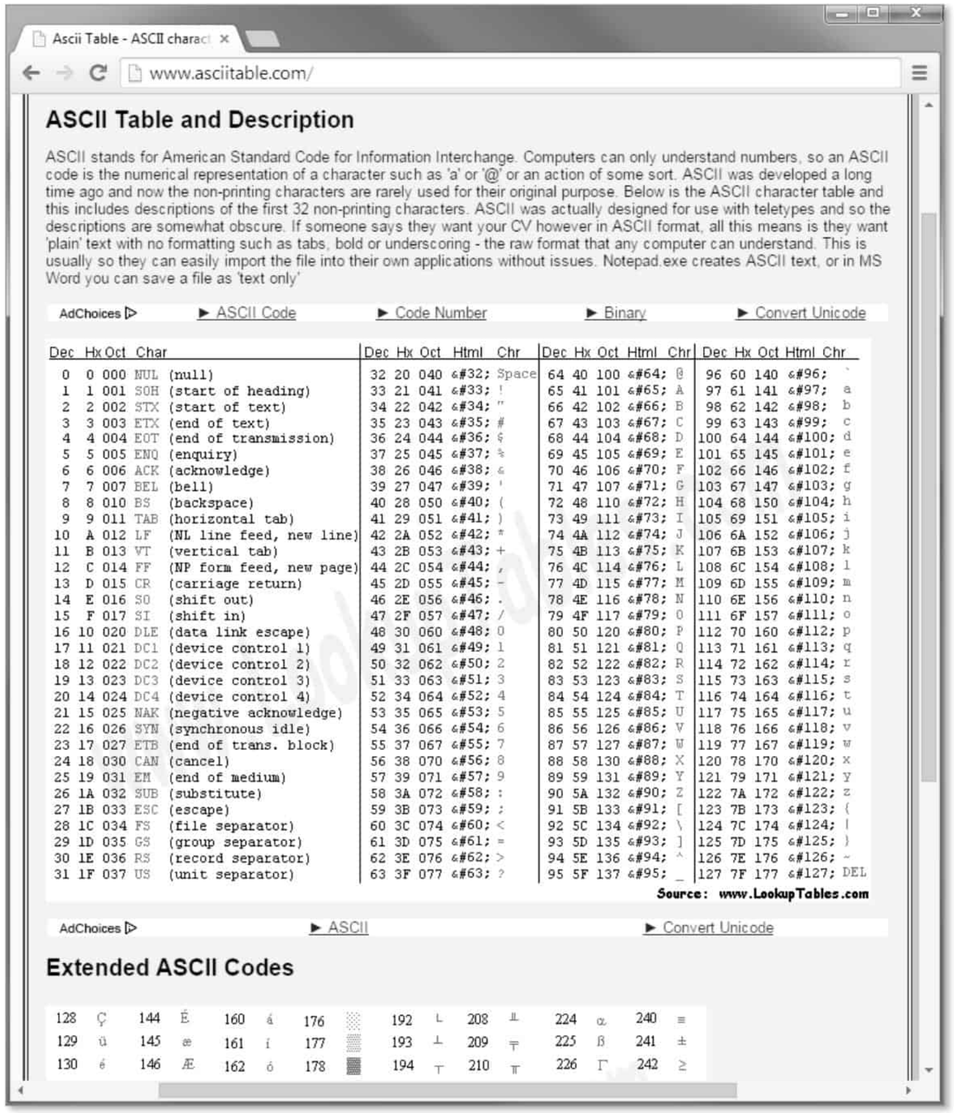


Abbildung 5.2 Eine ASCII-Tabelle zeigt den Code der einzelnen Zeichen (http://www.asciitable.com).

Sie müssen nun aber nicht die komplette ASCII-Tabelle auswendig können, um das Ergebnis eines String-Vergleichs vorauszusagen. Einige Merkregeln helfen:

- Kleinbuchstaben sind immer größer als Großbuchstaben, da sie höhere ASCII-Codes haben.
- Die Großbuchstaben haben die ASCII-Codes von 65 bis 90 in alphabetischer Reihenfolge.
- Die Kleinbuchstaben haben die Codes von 97 bis 121 in alphabetischer Reihenfolge.
- Die Buchstaben von Strings werden von links nach rechts miteinander verglichen.

Einige Beispiele verdeutlichen die Regeln:

```
$a = "a";
$b = "b";
$erg = $a < $b;
```

Mit den obigen Zeilen erhält die Variable `$erg` den Wert `true`, da das kleine `a` einen niedrigeren ASCII-Code hat als das kleine `b`.

Das nächste Beispiel ergibt dagegen `false`:

```
$a = "a";
$b = "B";
$erg = $a < $b;
```

Der Grund: Das große `B` hat einen niedrigeren ASCII-Wert als alle Kleinbuchstaben, also auch als das kleine `a`.

Bei längeren Zeichenketten vergleicht PHP von links nach rechts:

```
$a = "abzzz";
$b = "acaaa";
$erg = $a < $b;
```

In diesem Beispiel ist demnach das Ergebnis `true`. Der Interpreter sieht, dass die erste Stelle gleich ist, prüft die zweite und merkt, dass dort das kleine `b` kleiner ist als das kleine `c`. Die Stellen danach spielen keine Rolle mehr.

Wenn zwei Zeichenketten unterschiedlich lang sind, erfolgt der Vergleich dennoch von links nach rechts: `Z` ist also größer als `Abend`. Sind die bei beiden Strings vorhandenen Zeichen gleich, ist die längere Zeichenkette immer größer:

```
$a = "abc";
$b = "abcde";
$erg = $a < $b;
```

In diesem Fall ist also `$b` größer als `$a` und das Ergebnis (`$erg`) deshalb `true`.

Hinweis

In den Arbeitsdateien im Ordner zu diesem Kapitel finden Sie die Datei *strings_vergleichen.php*, die die hier gezeigten Beispiele enthält.

Tipp

In PHP gibt es beim String-Vergleich ein Verhalten, das etwas außergewöhnlich ist. Werden zwei Strings miteinander verglichen, die numerische Zahlen enthalten, werden die Strings vor dem Vergleich in Zahlen umgewandelt.

```
$a = "5.40";
$b = "5.4";
$erg = $a == $b;
```

ergibt also `true`. Derselbe Vergleich mit exakter Gleichheit (`===`) ergäbe hingegen `false`.

Strings alphabetisch sortieren | Um Strings alphabetisch zu sortieren, gibt es einen einfachen Trick: Sie speichern die Strings in Variablen speziell für den Vergleich und wandeln sie in Klein- oder Großbuchstaben um, bevor Sie sie vergleichen.

```
$a = "a";
$b = "B";

$a_low = strtolower($a);
$b_low = strtolower($b);

if ($a_low < $b_low) {
  echo "$a liegt im Alphabet vor $b";
} else {
  echo "$b liegt im Alphabet vor $a";
}
```

Listing 5.3 Strings alphabetisch sortieren (»strings_sortieren.php«)



Abbildung 5.3 Nun werden Strings unabhängig von Groß- und Kleinschreibung verglichen.

Dieser einfache Trick kommt in der Praxis häufig in Verbindung mit Arrays zum Einsatz. Für das Sortieren eines Arrays bietet PHP eine eigene Funktion namens `sort()`.

```
$sammlung = array("Monet", "Chagal", "Dali", "Manet");
sort($sammlung);
```

Wenn Sie das erste und das letzte Element des sortierten Arrays ausgeben, erfolgt die korrekte Ausgabe `Chagal` und `Monet`.

```
echo "$sammlung[0] und $sammlung[3]";
```

Sobald allerdings eines der Elemente des Arrays mit Kleinbuchstaben beginnt, scheitert die Sortierung mit `sort()`.

```
$sammlung = array("Monet", "chagal", "Dali", "Manet");
```



Abbildung 5.4 Die alphabetische Reihenfolge stimmt nicht.

Als Lösung kombinieren wir den Trick zum korrekten Sortieren mit der Funktion `usort(Array, Sortierfunktion)`, die eine eigene Sortierfunktion erlaubt. Die Sortierfunktion vergleicht immer zwei Elemente des Arrays und liefert als Ergebnis des Vergleichs entweder `0` (gleich), `1` (Parameter a größer als b) oder `-1` (Parameter b größer als a):

```
<?php
$sammlung = array("Monet", "chagal", "Dali", "Manet");

usort($sammlung, "sortieren");

function sortieren($a, $b) {
  $a_low = strtolower($a);
  $b_low = strtolower($b);
```



```
  if ($a_low == $b_low) {
    return 0;
  } elseif ($a_low > $b_low) {
    return 1;
  } else {
    return -1;
  }
}
echo "$sammlung[0] und $sammlung[3]";
?>
```

Listing 5.4 Sortieren mit einer Funktion (»strings_sortieren_funktion.php«)

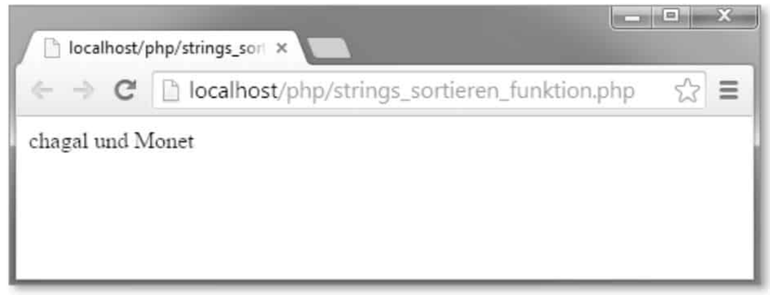


Abbildung 5.5 Nun funktioniert die alphabetische Sortierung trotz Kleinbuchstaben.

Die nervige 0 | Bei String-Vergleichen ist die 0 etwas problematisch. Ein String, der nicht mit einer Zahl beginnt, ist im direkten Vergleich immer gleich einer 0:

```
$a = "a";
if ($a == 0) {
  echo "$a hat den Wert 0";
}
```



Abbildung 5.6 a gleich 0?

Damit ein String nicht mehr gleich 0 ist, müssen Sie auf exakte Gleichheit prüfen:

```
if ($a === 0) {
  echo "$a hat den Wert 0";
}
```

5.1.3 Logische Operatoren

Ein Vergleich mit einem Vergleichsoperator liefert einen Wahrheitswert. Die Betonung liegt auf *einen*. Wenn Sie mehrere Vergleiche oder mehrere Wahrheitswerte verbinden möchten, benötigen Sie *logische Operatoren*.

Mit zwei Wahrheitswerten sieht das so aus:

```
  true     &&     false
Operand Operator Operand
```

In diesem Fall kommt das logische UND zum Einsatz (`&&` oder alternativ `and`). Es liefert nur dann `true`, wenn beide Operanden `true` liefern. Deswegen ergibt die obige Zeile `false`.

Wenn Sie zwei Vergleiche einsetzen, könnten Sie einen logischen Operator etwa so verwenden:

```
$erg = 7 > 3 && 2 < 4;
```

Diese Zeile liefert als Ergebnis `true`, da beide Vergleiche `true` liefern und entsprechend der logische Vergleich `true` ergibt.

Hinweis

Sie können auch mehrere logische Operationen kombinieren. Empfehlenswert ist dann allerdings aus Gründen der Übersichtlichkeit der Einsatz von Klammern.

Die folgende Tabelle gibt eine Übersicht über die logischen Operatoren:

Operator	Beispiel	Beschreibung
`&& and`	`7 > 3 && 2 < 4; //true`	Logisches UND. Liefert `true`, wenn beide Operanden `true` liefern.
`|| or`	`7 < 3 || 2 < 4; //true`	Logisches ODER. Liefert `true`, wenn einer der beiden oder beide Operanden `true` liefern.

Tabelle 5.4 Die logischen Operatoren

Operator	Beispiel	Beschreibung
`xor`	`7 > 3 xor 2 < 4; //false`	Logisches ENTWEDER ODER. Liefert nur `true`, wenn einer der beiden Operanden `true` ist. Liefert `false`, wenn keiner der beiden oder beide Operanden `true` sind.
`!`	`!false; //true`	Negation. Kehrt einen Wahrheitswert um. Aus `true` wird `false` und aus `false` `true`.

Tabelle 5.4 Die logischen Operatoren (Forts.)

Hinweis

Das logische ODER entspricht nicht dem »oder« im deutschen Sprachgebrauch. Das deutsche »oder« steht für »entweder – oder«, während beim logischen ODER auch beide Alternativen eintreffen können. Das deutsche »entweder – oder« ist in PHP in `xor` abgebildet, kommt in der Programmierung allerdings recht selten zum Einsatz.

Für logisches UND und logisches ODER gibt es jeweils zwei Varianten in PHP: eine mit Symbolen und eine mit Buchstaben. Der einzige Unterschied besteht darin, dass die Variante mit Symbolen eine höhere Operatorpriorität besitzt (siehe Abschnitt 5.1.6, »Rangfolge der Operatoren«).

Short-circuit evaluation

Die *short-circuit evaluation (Kurzschlussauswertung)* ist ein Programmierkonzept von PHP, das die Performance erhöhen soll. Wenn ein Vergleich mit logischem Operator schon beim ersten Operanden erfüllt ist oder scheitert, wird der zweite Operand nicht mehr geprüft. Im folgenden Beispiel liefert schon der erste Vergleich `false`. Da das logische UND damit schon `false` ergibt, wird der zweite Vergleich nicht mehr überprüft.

```
7 < 3 && 2 < 4;
```

In der Praxis hat dieses Verhalten von PHP meist keine Auswirkungen auf Ihre Programmierung. Eine Ausnahme sind Funktionsaufrufe. Dort macht es dann Sinn, zuerst einfache Vergleiche durchzuführen.

5.1.4 Binärzahlen und bitweise Operatoren

Die *bitweisen Operatoren* kommen eher selten zum Einsatz. Sie dienen dazu, direkt auf Bitebene zu arbeiten. Bevor Sie aber die Operatoren näher kennenlernen, erfahren Sie die Grundlagen zur Bitabbildung von Zahlen.

Ein Bit nimmt die Werte 0 oder 1 an. Da es zwei mögliche Werte gibt, heißt dies auch Binärwert. Die zugehörige Schreibweise für Daten ist die Binärschreibweise. Jede Ganzzahl (Integer) lässt sich in Bits schreiben.

Die binäre Schreibweise besteht aus einem Muster. Das Muster hat so viele Stellen, wie die Zahl Bits besitzt. Eine Zahl mit 4 Bit hat also vier Stellen und kann 2^4 Zahlen, also 16 Zahlen, darstellen:

0010

steht für die Zahl 2. Das Bitmuster wird von rechts nach links gelesen. Die rechte Zahl steht für die 1, die zweite von rechts für die 2, die dritte für die 4, die vierte für 8, die fünfte für ... jetzt wird es offensichtlich. Diese Zahlen werden addiert und ergeben die Ganzzahl.

Sehen Sie sich einige Beispiele an:

1111
steht für 8 + 4 + 2+ 1 gleich 15.

1010
steht für 8 + 0 + 2 + 0 gleich 10.

Hinweis

Seit PHP 5.4 können Zahlen auch direkt in der Binärzahl-Schreibweise in PHP eingesetzt werden. Hierzu wird der Zahl ein `0b` vorangesetzt, z. B. `0b1111` für 15.

Die bitweisen Operatoren arbeiten mit Ganzzahlen als binären Mustern. Das bitweise UND (&) setzt überall dort eine 1, wo beide Operanden eine 1 besitzen:

```
10 & 3
```

wird also von PHP intern erst in das binäre Muster[3] umgewandelt:

```
1010 & 0011
```

Ab PHP 5.4 können Sie auch direkt schreiben:

```
0b1010 & 0b0011
```

Der Vergleich gibt an der ersten Stelle von rechts nicht bei beiden Operanden eine 1. Deswegen erhält die Zahl, die aus dem Vergleich resultiert, an dieser Stelle eine 0. An der zweiten Stelle besitzen beide Operanden eine 1, insofern steht dort im Ergebnis eine 1. Führt man dies weiter, entsteht folgendes binäres Muster:

3 Binäre Muster lassen sich in PHP nicht direkt angeben. Sie kommen hier nur für verbessertes Verständnis zum Einsatz. In PHP müssen Sie immer Integer angeben.


```
0b0010
```

Es entspricht der Ganzzahl 2.

Die folgende Tabelle zeigt alle bitweisen Operatoren mit einfachen Beispielen in binärem Muster. Um die Beispiele in PHP umzusetzen, müssen Sie die binären Muster als Ganzzahlen schreiben.

Operator	Beispiel	Beschreibung
&	`1010 & 0011 //Erg: 0010 = 2`	Bitweises UND; schreibt an die Stellen eine 1, an denen **beide** Operanden eine 1 besitzen.
|	`1010 | 0011 //Erg: 1011 = 11`	Bitweises ODER; schreibt an die Stellen eine 1, an denen bei **einem** oder **beiden** Operanden 1 steht.
^	`1010 ^ 0011 //Erg: 1001 = 9`	Bitweises ENTWEDER ODER; schreibt an die Bits eine 1, an denen nur **einer der beiden** Operanden eine 1 besitzt.
~	`~1010 //Erg: 0101 = -11`	Bitweise Negation; wandelt eine 0 in eine 1 und eine 1 in eine 0 um. Allerdings ist die Basis ein Integer mit 32 Bit (signed – sprich mit Vorzeichen), weswegen das Ergebnis keine direkte Umkehrung des Werts ist.
<<	`1010 << 1 //Erg: 10100 = 20`	Bitweise Verschiebung nach links verschiebt das Binärmuster des linken Operanden um die im rechten Operanden angegebenen Stellen nach links. Die rechte Seite wird durch Nullen aufgefüllt. Die Verschiebung um eine Stelle entspricht der Multiplikation mit 2, um zwei Stellen der Multiplikation mit 4, der um drei der Multiplikation mit 8 usw.
>>	`1010 >> 1 //Erg: 0101 = 5`	Bitweise Verschiebung nach rechts um die vom rechten Operanden angegebenen Stellen. Die Bits, die rechts übrig bleiben, werden gelöscht. Hat der linke Operand ein negatives Vorzeichen, wird die linke Seite mit Einsen aufgefüllt; ansonsten mit Nullen. Das Verschieben um ein Bit entspricht der Division durch 2 (ohne Rest), das um zwei der Division durch 4, das um vier der Division durch 8 usw.

Tabelle 5.5 Die bitweisen Operatoren

Hinweis

Die bitweise Verschiebung wird von den bitweisen Operatoren mit am häufigsten eingesetzt, da sie eine einfache Möglichkeit zur Division und Multiplikation mit Zweierpotenzen bietet. Ansonsten sind bitweise Operatoren in der Webentwicklung allerdings nicht sehr häufig.

Alle binären Operatoren gibt es auch in der Kurzform mit dem Zuweisungsoperator. Die folgenden Zeilen ändern den Wert der Variablen `$a` von `10` auf `40`:

```
$a = 10;
  $a <<= 2;
```

Binäres Muster erzeugen

Die Umwandlung einer Ganzzahl in die binäre Schreibweise ist mit Stift und Papier nicht ganz trivial. Es bietet sich also an, in PHP einen kleinen Konverter zu schreiben. Der hier beschriebene Konverter soll die Zahlen von 0 bis 255 (2^8) in die binäre Schreibweise mit 8 Stellen umwandeln.

Hinweis

Viele hier verwendete Möglichkeiten kennen Sie bisher noch nicht, wenn Sie neu in PHP sind. Die Programmierkonstrukte lernen Sie noch in diesem Kapitel kennen, den Umgang mit Formularen finden Sie in Kapitel 14, »Formulare«.

Die Umwandlung der in das Formular eingegebenen Zahl besteht aus drei wichtigen Elementen:

- Eine Schleife durchläuft die 8 Stellen der binären Zahl von 7 bis 0. Der Zähler `$i` dient in den folgenden Anweisungen sowohl zum Zugriff auf das Array als auch für die Berechnung.

```
for ($i = 7; $i >= 0; $i--) {
  //Anweisungen
}
```

- Innerhalb der Schleife überprüfen Sie mit dem bitweisen UND, ob bei der Zahl an der Stelle eine 1 steht. Die zweite Potenz des Zählers ergibt die jeweilige Stelle. Im ersten Durchlauf $2^7 = 128$, im zweiten $2^6 = 64$ usw. Steht eine 1 an der jeweiligen Stelle, wird im Array eine `1` gesetzt; ansonsten eine `0`.

```
if ($zahl & pow(2, $i)) {
  $binaerwerte[$i] = 1;
} else {
  $binaerwerte[$i] = 0;
}
```

- Das Array mit den Datenwerten wandeln Sie zum Schluss mit `join()` in einen String um:

```
$binaer = join("", $binaerwerte);
```


Im Folgenden finden Sie das komplette Skript abgedruckt:

```
<?php
  $zahl = "";
  $binaer = "";
  if ($_GET["Senden"] == "Umwandeln") {
    $zahl = $_GET["eingabe"];
    $binaerwerte = array();

    for ($i = 7; $i >= 0; $i--) {
      if ($zahl & pow(2, $i)) {
        $binaerwerte[$i] = 1;
      } else {
        $binaerwerte[$i] = 0;
      }
    }
    $binaer = join("", $binaerwerte);
  }

?>

<html>
  <head>
    <title>Bin&auml;r</title>
  </head>
  <body>
    <form>
      <input type="text" name="eingabe" value="<?=$zahl ?>" />
      <input type="text" name="ausgabe" value="<?=$binaer ?>" />
      <input type="submit" name="Senden" value="Umwandeln" />
    </form>
  </body>
</html>
```

Listing 5.5 Die Umwandlung in binäre Schreibweise (»bitweise_umwandeln.php«)



Abbildung 5.7 Die Zahl 20 verwandelt sich in die binäre Schreibweise.

5.1.5 Operatoren, die aus der Reihe tanzen

Den Zuweisungsoperator, das Gleichheitszeichen, haben Sie schon in Abschnitt 4.3, »Variablen«, im Einsatz gesehen. Er dient zum Zuweisen von Werten zu Variablen, fällt aber auch in die Kategorie Operator. Neben diesem gibt es noch einige andere Operatoren, die oftmals nicht als Operatoren bekannt sind.

Hinweis

Einige dieser Exoten finden Sie hier, die Operatoren, die für Objekte relevant sind, werden in Kapitel 11, »Objektorientiert programmieren«, besprochen.

Fehlerunterdrückung

In die Kategorie der ungewöhnlichen Operatoren fällt der Operator zur Unterdrückung von Fehlermeldungen, das @-Symbol. Wenn Sie diesen Operator vor einen Ausdruck setzen, wird eine von diesem Ausdruck erzeugte Fehlermeldung unterdrückt. Ein Ausdruck kann ein Funktionsaufruf, das Laden eines externen Skripts oder Ähnliches sein.

```
@funktion();
```

unterdrückt beispielsweise eine Fehlermeldung auch dann, wenn eine Funktion mit dem Namen `funktion()` nicht vorhanden ist.

Hinweis

Die Wirkung von @ bei Ausdrücken ist sehr stark. Da die Fehlermeldung unterdrückt wird, fällt die Fehlersuche bei der Verwendung von @ schwer. Deswegen sollten Sie @ in der Praxis sehr vorsichtig einsetzen und es beim Testen zuerst entfernen, wenn das Skript nicht nach Wunsch funktioniert.

Hinweis

Mehr zum Aufspüren von Fehlern erfahren Sie in Kapitel 36, »Fehlersuche und Debugging«.

Shell-Operator

Der Shell-Operator dient dazu, einen Befehl in der Shell auszuführen. Der Befehl steht dabei in von links oben nach rechts unten geneigten Strichen, die auch *Backticks* genannt werden (`` `Anweisung` ``).

»print«

Das Sprachkonstrukt `print` gilt in PHP auch als Operator. Dies hat eigentlich nur eine Auswirkung: `print` kommt in der Reihenfolge der Operatoren vor und hat dort einen höheren Rang als das logische UND mit `and`. Das heißt, die folgende Zeile gibt `1` für `true` aus, da zuerst die Ausgabe des ersten Vergleichs erfolgt, bevor der logische Operator zum Einsatz kommt:

```
print 7 > 3 and 7 < 3;
```

Würden Sie stattdessen das logische UND mit Symbolen verwenden, würde kein Wert, also `false`, ausgegeben, da das logische UND mit Symbolen einen höheren Rang als `print` besitzt.

```
print 7 > 3 && 7 < 3;
```

`echo` ist im Gegensatz zu `print` kein Operator. Sehen Sie sich den Unterschied an:

```
echo 7 > 3 and 7 < 3;
```

Mit `print` hätte die obige Zeile `1` für `true` ausgegeben, mit `echo` wird kein Wert, also `false`, ausgegeben.

Konditionaler Operator

Der konditionale Operator dient dazu, zwischen zwei Ausdrücken zu wählen. Wenn die `Bedingung` eintritt, wird `Ausdruck1` verwendet, ansonsten `Ausdruck2`. Der verwendete Ausdruck liefert einen Wert zurück.

```
Bedingung ? Ausdruck1 : Ausdruck2;
```

Da der konditionale Operator mit `Bedingung`, `Ausdruck1` und `Ausdruck2` als einziger Operator in PHP drei Operanden hat, heißt er auch *ternärer Operator*.

Der folgende Code prüft, ob eine Variable den Wert 4 hat, und liefert entsprechend eine Rückgabe.

```
$a = 4;
$erg = $a != 4 ? 4 : 8;
```

Die Variable $erg hat nach dem Einsatz des konditionalen Operators den Wert 8.

Hinweis

Wenn Sie als Ausdruck eine Anweisung ausführen, ersetzt der konditionale Operator eine einfache Fallunterscheidung. Dies gilt allerdings als eher unsaubere Programmierung. Sie sollten den konditionalen Operator also wirklich nur dann wählen, wenn Sie zwischen zwei Ausdrücken wählen möchten.

Eine Neuerung gibt es in PHP 5.3 beim ternären Operator: Der Mittelteil kann ausgelassen werden.

```
Bedingung ?: Ausdruck2;
```

In diesem Fall wird direkt der Wert der Bedingung zurückgegeben, außer die Bedingung ist nicht erfüllt. In diesem Fall wird Ausdruck2 zurückgeliefert. Im folgenden Beispiel wird zuerst die Bedingung, hier verkörpert durch die Variable $a, geprüft.

```
$a = false;
$erg = $a ?: 'false';
```

Da $a false ist, erhält $erg den String false.

5.1.6 Rangfolge der Operatoren

Wenn eine Anweisung aus mehreren Operatoren besteht, muss PHP wissen, in welcher Reihenfolge die Operationen ausgeführt werden sollen. Werfen Sie einen Blick auf die folgende Zeile:

```
$erg = 2 + 4 * 5;
```

Das Ergebnis ist 22. Zuerst wird die Multiplikation ausgeführt, dann 2 addiert. Der Operator * hat also einen höheren Rang[4] als das +. Dies entspricht in der Mathematik der Regel »Punkt vor Strich«.

Sie könnten die Ausführreihenfolge natürlich auch beeinflussen. Hierzu verwenden Sie runde Klammern:

```
$erg = (2 + 4) * 5;
```

4 Der Begriff Rang wird oft auch als Reihenfolge oder Präferenz bezeichnet.

Das Ergebnis dieser Zeile ist 30, da zuerst 2 und 4 addiert werden und die Summe dann mit 5 multipliziert wird. Die runden Klammern sind selbst ein Operator.

Da nicht alle Rangfolgen so einleuchtend sind wie die einfachen mathematischen Regeln, finden Sie in Tabelle 5.6 die Operatoren mit dem jeweiligen Rang. Je höher der Rang, desto höher die Präferenz. Das heißt, zuerst werden Operationen mit höherem Rang ausgeführt. Innerhalb eines Rangs richtet sich die Ausführreihenfolge nach der *Assoziativität*. Sie gibt für alle Operatoren, die mehrmals hintereinanderstehen können, an, ob die Operatorenrangfolge von links nach rechts oder von rechts nach links verläuft. Im Fall von Multiplikation, Division und Modulo wird also zuerst die Operation links, dann die rechts daneben und so weiter ausgeführt.

```
$erg = 6 / 3 * 2;
```

ergibt also 4 und nicht 1.

Rang	Assoziativität	Operator
20	ohne	new
19	rechts	[]
18	rechts	! ~ ++ -- (Typoperatoren) @
17	links	* / %
16	links	+ - .
15	links	<< >>
14	ohne	< <= > >=
13	ohne	== != === !==
12	links	&
11	links	^
10	links	|
9	links	&&
8	links	||
7	links	? :
6	rechts	= += -= *= /= .= %= &= |= ^= <<= >>=
5	rechts	print

Tabelle 5.6 Die Reihenfolge der Operatoren

Rang	Assoziativität	Operator
4	links	and
3	links	xor
2	links	or
1	links	,

Tabelle 5.6 Die Reihenfolge der Operatoren (Forts.)

5.2 Fallunterscheidungen

Links oder rechts? Diese einfache Frage beschäftigt den Programmierer nicht nur an der Straßenkreuzung, sondern auch in seiner Webanwendung. Alles, was mit einem »Wenn« beginnen könnte, schreit geradezu nach einer Fallunterscheidung. Wenn der Nutzer »XY« eingibt, tue dies, wenn er dagegen »AB« eingibt, tue das.

PHP bietet für diese grundlegenden Überprüfungen und Entscheidungen zwei Programmierkonstrukte: zum einen die `if`-Fallunterscheidung, die fast in jeder aktuellen Programmiersprache anzutreffen ist, und zum anderen `switch case`.

5.2.1 »if«

Die `if`-Fallunterscheidung besteht in ihrer Grundform aus zwei wichtigen Elementen: einer Bedingung, die überprüft wird, und einem Anweisungsblock, der nur ausgeführt wird, wenn die Bedingung erfüllt ist. Damit der PHP-Interpreter etwas damit anfangen kann, müssen Sie sich an die einfache Syntax halten:

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

Ins Deutsche übersetzt bedeutet das:

- Wenn Bedingung erfüllt,
- führe Anweisungen aus.

Sollte die Bedingung nicht erfüllt sein, werden die Anweisungen ignoriert. Anschließend wird – in beiden Fällen – der Code nach der `if`-Fallunterscheidung ausgeführt. Die Anweisungen innerhalb der geschweiften Klammern heißen auch *Anweisungsblock*.

Das folgende Beispiel überprüft das Alter eines Kindes. Das Alter wird hier über die Variable `$alter` im Quellcode angegeben. Natürlich kann es sich dabei auch um eine Benutzereingabe in ein Formular oder einen Wert aus einer Datenbank handeln.

```
<?php
  $alter = 4;
  if ($alter > 3) {
    echo "Mit $alter Jahren ist das Kind dem Säuglingsalter entwachsen.";
  }
?>
```

Listing 5.6 »if«-Fallunterscheidung (»if.php«)

Was, denken Sie, wird ausgegeben? Richtig, der Text mit dem Alter des Kindes (siehe Abbildung 5.8).



Abbildung 5.8 Das Kind ist über 3 Jahre alt.

Wenn Sie den Wert der Variablen `$alter` beispielsweise auf `3` oder eine niedrigere Zahl ändern, erfolgt keine Ausgabe. Die Seite bleibt also leer, weil die Bedingung nicht erfüllt ist und dementsprechend die Ausgabeanweisung mit `echo` überhaupt nicht ausgeführt wird.

»elseif«

In der Praxis gibt es häufig nicht nur eine Alternative, sondern mehrere. Eine mögliche Lösung besteht darin, einfach mehrere `if`-Fallunterscheidungen hintereinanderzuschreiben.

```
if ($alter > 3) {
  echo "Mit $alter Jahren ist das Kind dem Säuglingsalter entwachsen.";
}
if ($alter >= 2) {
  echo "Das $alter Jahre alte Baby kann ein wenig sprechen.";
}
```

Was passiert, wenn die Variable `$alter` den Wert `6` besitzt? Da die beiden `if`-Fallunterscheidungen überhaupt nichts miteinander zu tun haben, werden beide getrennt geprüft. Da beide Bedingungen erfüllt sind, führt PHP beide Anweisungen aus (siehe Abbildung 5.9).



Abbildung 5.9 Ein 6 Jahre altes Baby, das nur ein wenig sprechen kann?

Um mehrere Bedingungen in einer Fallunterscheidung zu überprüfen, gibt es `elseif`.

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

Der Anweisungsblock von `elseif` wird nur dann ausgeführt, wenn die `if`-Bedingung nicht erfüllt war und die Bedingung von `elseif` erfüllt ist. Verwenden Sie statt der zwei `if`-Anweisungen für das letzte Beispiel `elseif`:

```
$alter = 6;
if ($alter > 3) {
  echo "Mit $alter Jahren ist das Kind dem Säuglingsalter entwachsen.";
} elseif ($alter >= 2) {
  echo "Das $alter Jahre alte Baby kann ein wenig sprechen.";
}
```

Listing 5.7 Alternativen mit »elseif« prüfen (»elseif.php«)

In diesem Beispiel wird zuerst die `if`-Bedingung überprüft. Da sie erfüllt ist, wird der Anweisungsblock ausgeführt. Dann verlässt PHP die Fallunterscheidung. Die `elseif`-Bedingung wird also gar nicht mehr überprüft. Das Ergebnis sehen Sie in Abbildung 5.10.



Abbildung 5.10 Nur noch der »if«-Anweisungsblock wird ausgeführt.

Hinweis

Sie können beliebig viele `elseif`-Bedingungen hintereinander verwenden. Sobald die erste Bedingung erfüllt ist, wird die Fallunterscheidung verlassen.

»else«

Mit `if` und `elseif` können Sie viele Fälle abdecken, oftmals aber nicht alle. Deswegen gibt es den `else`-Anweisungsblock.

```
if (Bedingung) {
  Anweisungen;
} elseif (Bedingung) {
  Anweisungen;
} else {
  Anweisungen;
}
```

Die `else`-Anweisungen werden immer dann ausgeführt, wenn keine der vorherigen Bedingungen erfüllt ist. Im folgenden Beispiel treffen die `if`- und die `elseif`-Bedingung nicht zu. Deswegen erfolgt die Ausgabe aus dem `else`-Anweisungsblock:

```
$alter = 18;
if ($alter > 3 && $alter < 18) {
  echo "Mit $alter Jahren ist das Kind dem Säuglingsalter entwachsen.";
} elseif ($alter >= 2 && $alter <= 3) {
  echo "Das $alter Jahre alte Baby kann ein wenig sprechen.";
} else {
  echo "Noch sehr kleines Baby oder schon erwachsen.";
}
```

Listing 5.8 Die »else«-Anweisung (»else.php«)



Abbildung 5.11 Die »else«-Anweisung wird ausgeführt, da keine der vorigen Bedingungen zutrifft.

Hinweis

Die `elseif`-Anweisung ist eigentlich eine Kombination aus `if` und `else`, die Programmierer erfunden haben, um sich das Leben zu erleichtern. Nur mit `if` und `else` könnten Sie `elseif` so nachbilden:

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

Kurzformen

Die `if`-Fallunterscheidung lässt sich auch kürzer schreiben, indem Sie alles in eine Zeile packen:

```
if (Bedingung) { Anweisung; }
```

Wenn im Anweisungsblock nur eine Anweisung vorkommt, können Sie die geschweiften Klammern einfach weglassen:

```
if (Bedingung)
  Anweisung;
```

Sie können das Ganze dann auch in eine Zeile schreiben:

```
if (Bedingung) Anweisung;
```


Und natürlich funktioniert die Kurzform auch mit `elseif` und `else`:

```
$alter = 0;
if ($alter > 3 && $alter < 18) echo "Jugend";
elseif ($alter >= 2 && $alter <= 3) echo "Sprechalter";
else echo "Kleines Baby oder Erwachsener";
```

Listing 5.9 Fallunterscheidung in Kurzform (»if_kurzform.php«)

Hinweis

Diese Fallunterscheidung in drei Zeilen spart zwar ein wenig Tipparbeit, kann allerdings im Nachhinein zu Problemen führen, da sie recht schlecht lesbar ist. Wenn Sie Ihren Code nach einem Monat wieder ansehen, benötigen Sie erst mal einige Zeit, um kryptische Fallunterscheidungen zu entwirren. Und der Kollege, der mit Ihrem Code weiterarbeiten muss, hat auch Mühe mit dieser unübersichtlichen Variante.

Alternative Form

Mit den Kurzformen sind Sie noch nicht am Ende der alternativen Schreibweisen für eine an sich einfache Fallunterscheidung angelangt. PHP bietet auch noch eine Schreibweise mit Doppelpunkt und `endif`:

```
if (Bedingung) :
  Anweisungen;
elseif (Bedingung) :
  Anweisungen;
else:
  Anweisungen;
endif;
```

Diese Syntax erinnert ein wenig an Visual Basic. Sie ist in PHP eigentlich ungebräuchlich, hat aber ein praktisches Anwendungsgebiet: die einfache Ausgabe von HTML-Code.[5] Und so sieht das in der Praxis aus:

```
<?php
$a = 10;
if ($a < 8) :
?>

<p>if-Bedingung erf&uuml;llt<p>
```

5 Dies funktioniert allerdings auch mit geschweiften Klammern, zumindest in neueren Versionen von PHP 4 und in PHP 5. Allerdings wird die Doppelpunkt-Notation oft noch vorgezogen, da sie etwas übersichtlicher und schon so lange gebräuchlich ist.

```
<?php
elseif ($a >= 8 && $a < 20) :
?>

<p>elseif-Bedingung erf&uuml;llt<p>

<?php
else:
?>

<p>else-Fall eingetreten<p>

<?php
endif;
?>

<p>HTML au&szlig;erhalb der Fallunterscheidung</p>
```

Listing 5.10 Die HTML-Ausgabe ist in die Fallunterscheidung eingeflochten (»if_alternativeForm.php«).



Abbildung 5.12 In diesem Fall tritt die »elseif«-Bedingung ein.

Verschachtelt

Sie können `if`-Fallunterscheidungen beliebig ineinander verschachteln. Einzige Bedingung ist, dass Sie sich noch in Ihrem Gedankengewirr zurechtfinden. Das folgende Beispiel zeigt eine verschachtelte Fallunterscheidung, die gleichzeitig auch die Komplexität von Verschachtelungen zeigt:

```
$alter = 20;
if ($alter > 3) {
  echo "Mit $alter Jahren ist das Kind dem Säuglingsalter entwachsen.";
  if ($alter > 18) {
```



```
    if ($alter <= 21) {
      echo "Schon erwachsen?";
    } else {
      echo "Erwachsen";
    }
  } elseif ($alter >= 10) {
    echo "Ein Teenie";
  } else {
    echo "Ein kleines Kind";
  }
} else {
  echo "Noch ein Baby";
}
```

Listing 5.11 Verschachtelte »if«-Fallunterscheidungen (»if_verschachtelt.php«)



Abbildung 5.13 Die Ausgabe der verschachtelten »if«-Anweisungen

5.2.2 »switch«

Die zweite Fallunterscheidung in PHP ist `switch`. Auch sie ist in vielen anderen Programmiersprachen zu finden, manchmal heißt sie allerdings anders, in Visual Basic und VBScript beispielsweise `select`.

`switch` prüft für eine Variable oder einen Ausdruck in einzelnen Fällen (engl.: *case*) die Werte. Stimmt ein Wert mit dem Wert der Variablen überein, werden die folgenden Anweisungen ausgeführt. Die `break`-Anweisung verlässt anschließend die `switch`-Fallunterscheidung.

```
switch (Variable) {
  case Wert1:
    Anweisungen;
    break;
  case Wert2:
    Anweisungen;
```

```
    break;
  case Wert3:
    Anweisungen;
    break;
}
```

Die `switch`-Fallunterscheidung eignet sich vor allem, wenn Sie eine Variable auf verschiedene Werte überprüfen möchten. Im Folgenden sehen Sie ein Beispiel:

```
$alter = 30;
switch ($alter) {
  case 29:
    echo "Sie sind 29.";
    break;
  case 30:
    echo "Sie sind 30.";
    break;
  case 31:
    echo "Sie sind 31.";
    break;
}
```

Listing 5.12 Die »switch«-Fallunterscheidung (»switch.php«)

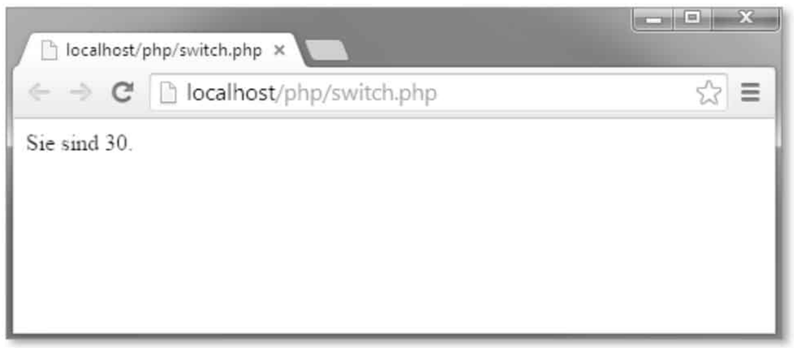


Abbildung 5.14 Der zweite Fall tritt ein.

»break«

Wenn Sie in einer `switch`-Fallunterscheidung die `break`-Anweisung weglassen, werden alle Anweisungen ab dem Fall, der zutrifft, ausgeführt. Dieses Verhalten kann manchmal gewünscht sein, wenn man beispielsweise für mehrere Fälle dieselben Anweisungen ausführen möchte, führt aber im folgenden Fall zu einem ungewollten Ergebnis:


```
$alter = 30;
switch ($alter) {
  case 29:
    echo "Sie sind 29.";
  case 30:
    echo "Sie sind 30.";
  case 31:
    echo "Sie sind 31.";
}
```

Listing 5.13 Die »switch«-Fallunterscheidung ohne »break« (»switch_ohne.php«)



Abbildung 5.15 Ohne »break« werden mehrere Anweisungen ausgeführt.

Der Hintergrund für dieses Verhalten ist schnell erklärt: `switch`-Fallunterscheidungen werden stur Zeile für Zeile ausgeführt. Ist eine Bedingung eingetreten, heißt das, dass alle nachfolgenden Zeilen ausgeführt werden sollen. Die `case`-Zeilen werden dabei ignoriert, alle normalen Anweisungen aber ausgeführt. Manchmal ist dieses Verhalten durchaus gewünscht, vor allem wenn Anweisungen ab einem bestimmten Punkt ausgeführt werden sollen.

Der Standardfall – »default«

Wenn alle Fälle nicht eintreten, gibt es für `switch` noch einen Standardfall. Er beginnt mit dem Schlüsselwort `default`. Die Anweisungen folgen nach einem Doppelpunkt:

```
switch (Variable) {
  case Wert1:
    Anweisungen;
    break;
  case Wert2:
    Anweisungen;
    break;
  default:
    Anweisungen;
}
```

Mit dem Standardfall können Sie alles abfangen, was nicht in den vorherigen Fällen berücksichtigt wurde. Hier ein einfaches Beispiel, bei dem der Standardfall für alle zuständig ist, die nicht zwischen 29 und 31 Jahre alt sind:

```
$alter = 32;
switch ($alter) {
  case 29:
    echo "Sie sind 29.";
    break;
  case 30:
    echo "Sie sind 30.";
    break;
  case 31:
    echo "Sie sind 31.";
    break;
  default:
    echo "Sie sind nicht zwischen 29 und 31.";
  }
```

Listing 5.14 »switch« mit »default«-Anweisung (»switch_default.php«)



Abbildung 5.16 Der Standardfall ist eingetreten.

Mit Bedingung

Bisher haben Sie mit `switch` nur Werte geprüft. `switch` erlaubt aber auch die Angabe von Bedingungen für die einzelnen Fälle. Sie können also beispielsweise das Alter in Kategorien abprüfen:

```
switch ($alter) {
  case $alter >= 10 && $alter < 30:
    echo "Sie sind zwischen 10 und 29.";
    break;
  case $alter >= 30 && $alter < 50:
    echo "Sie sind zwischen 30 und 49.";
    break;
  case $alter >= 50 && $alter < 70:
    echo "Sie sind zwischen 50 und 69.";
    break;
  default:
    echo "Sie passen in keine der Kategorien.";
}
```


Listing 5.15 »switch« mit Bedingung (»switch_bedingung.php«)

> **Hinweis**
>
> Zwecks besserer Übersichtlichkeit bietet es sich u. U. an, die Bedingungen in runde Klammern zu setzen. Erforderlich ist dies allerdings nicht.

Vergleich zwischen »if« und »switch«

Wann verwenden Sie `if` und wann besser `switch`? Für diese Frage gibt es keine pauschale Antwort. In der Praxis ist es heute so, dass `switch` hauptsächlich zum Durchprüfen von Werten zum Einsatz kommt. Bedingungen werden meist eher mit `if` geprüft.

Die Syntax von `switch` ist ein wenig gewöhnungsbedürftig. Sie ist zwar eigentlich leicht kürzer als eine `if`-Fallunterscheidung mit geschweiften Klammern, allerdings erfordert `switch` die `break`-Anweisung. Schließlich bleibt es eine Geschmacksfrage, für welche Variante Sie sich entscheiden.

5.3 Schleifen

Mit Schleifen führen Sie Anweisungen mehrmals hintereinander aus. PHP kennt vier Arten von Schleifen, die auch in anderen Programmiersprachen recht gebräuchlich sind.[6] Drei der vier lernen Sie jetzt kennen, die letzte, `foreach`, kommt vor allem mit Objekten und Arrays zum Einsatz. Sie ist Teil von Kapitel 11, »Objektorientiert programmieren«.

5.3.1 »for«

Die `for`-Schleife ist die komfortabelste aller Schleifen. Sie besitzt bereits drei Argumente, um das Schleifenverhalten zu steuern:

```
for (Startanweisung; Bedingung; Durchlaufanweisung) {
  Anweisungen;
}
```

6 Fallunterscheidungen und Schleifen werden auch als Kontrollstrukturen bezeichnet.

Und so funktioniert es:

1. Zu Beginn wird die Startanweisung einmal ausgeführt.
2. Dann überprüft PHP die Bedingung.
3. Trifft diese zu, führt der Interpreter die Anweisungen innerhalb der Schleife aus. Trifft sie nicht zu, wird die Schleife sofort verlassen.
4. Nach den Anweisungen im Anweisungsblock (geschweifte Klammern) wird die Durchlaufanweisung ausgeführt.
5. Anschließend überprüft PHP wieder die Bedingung.
6. Trifft sie zu, werden wiederum die Anweisungen im Anweisungsblock und dann die Durchlaufanweisung ausgeführt.
7. Und so weiter ...

Startanweisung, Bedingung und Durchlaufanweisung werden dazu verwendet, zu steuern, wie oft eine Schleife durchlaufen wird. Sie bilden zusammen den Schleifenzähler.

Sehen Sie sich den folgenden Code an:

```
for ($i = 0; $i < 10; $i++) {
  echo "$i<br />";
}
```

Listing 5.16 Die »for«-Schleife (»for.php«)

Hier wird die Variable `$i` in der Startanweisung mit dem Wert `0` initialisiert. Sie ist die Zählervariable. Die Bedingung ist, dass `$i` kleiner als `10` bleibt. Im Anweisungsblock werden `$i` und ein Zeilenumbruch ausgegeben. Die Durchlaufanweisung erhöht `$i` mittels Inkrement um `1`. Können Sie sich vorstellen, was die Schleife ausgibt? Richtig, die Zahlen von 0 bis 9 – in Abbildung 5.17 sehen Sie es.

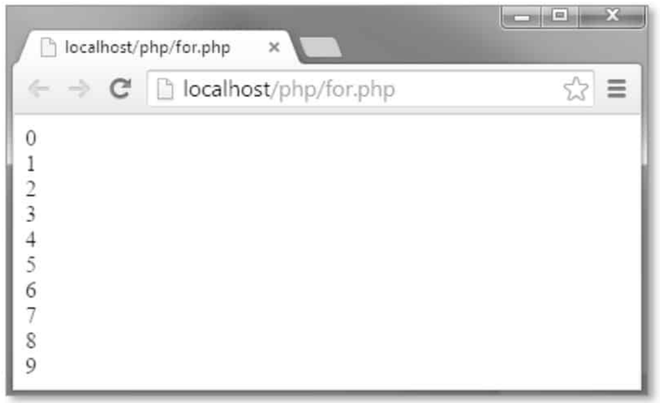


Abbildung 5.17 Die Zahlen von 0 bis 9

Hinweis

`$i`, `$j` etc. kommen oft als Zählervariablen zum Einsatz. Dies ist kein Muss, hat sich aber so eingebürgert.

Endlosschleifen

Die Durchlaufanweisung dient dazu, irgendwann dafür zu sorgen, dass die Bedingung nicht mehr erfüllt ist. Klappt dies nicht, ist die Bedingung also immer erfüllt, wird die Schleife endlos ausgeführt.

```
for ($i = 0; $i < 10; $i--) {
  echo "$i<br />";
}
```

Listing 5.17 Eine Endlosschleife (»for_endlos.php«)

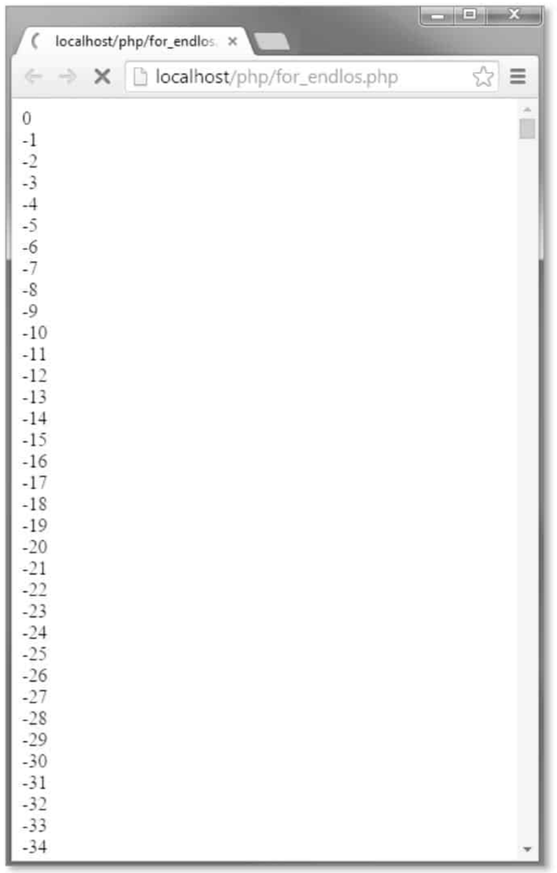


Abbildung 5.18 Die Endlosschleife läuft und läuft.

Eine solche Schleife heißt Endlosschleife. Im besten Fall sorgen Sie damit für große Rechnerlast auf Ihrem Server. Beim Testen ist dies nicht schlimm; da können Sie auch einfach auf die ABBRECHEN-Taste des Browsers klicken, um den Spuk zu beenden. Bei einem Produktivsystem kann eine Endlosschleife allerdings zu einigen Problemen führen, und die Ursache ist unter Umständen schwierig zu finden.

Andere Formen

Wie meist in PHP gibt es für die Syntax der `for`-Schleife noch einige alternative Lösungen, die ebenso funktionieren:

1. Wenn der Anweisungsblock nur aus einer Zeile besteht, können Sie analog zur if-Fallunterscheidung die geschweiften Klammern weglassen. Sie können sogar die Anweisung hinter die runden Klammern der Schleife packen.

```
for ($i = 0; $i < 10; $i++)
  echo "$i<br />";
```

Listing 5.18 »for« ohne geschweifte Klammern (»for_anders.php«)

2. Sie dürfen jedes der drei Argumente weglassen. Im folgenden Beispiel haben wir den Zähler vor der Schleife initialisiert und ändern ihn innerhalb des Anweisungsblocks. Dafür fallen Start- und Durchlaufanweisung weg. In diesem Fall arbeitet eine for-Schleife wie eine while-Schleife (siehe nächster Absatz).

```
$i = 0;
for (; $i < 10; ) {
  echo "$i<br />";
  $i++;
}
```

Listing 5.19 »for« ohne Start- und Durchlaufanweisung (»for_anders2.php«)

Hinweis

In der Praxis lassen Sie meist dann eines der Argumente weg, wenn es nicht gesetzt werden muss. Meist ist dies die Startanweisung, da die Variable oder das Element, das als Zähler dienen soll, schon vorher erzeugt wurde.

3. Sie können die for-Schleife wie die if-Fallunterscheidung mit einer Doppelpunkt-Syntax schreiben und dann mit endfor beenden. Dies erlaubt Ihnen, in der Schleife HTML-Code auszugeben.[7] Im folgenden Beispiel binden wir innerhalb des HTML-Codes noch einmal PHP ein, das die Zählervariable ausgibt:

7 Auch dies klappt ebenfalls – wie bei `if` – mit der Syntax mit geschweiften Klammern.


```
<?php
  for ($i = 0; $i < 10; $i++):
?>

<p>Ausgabe: <?=$i ?></p>

<?php
  endfor;
?>
```

Listing 5.20 »for« für die HTML-Ausgabe (»for_anders3.php«)

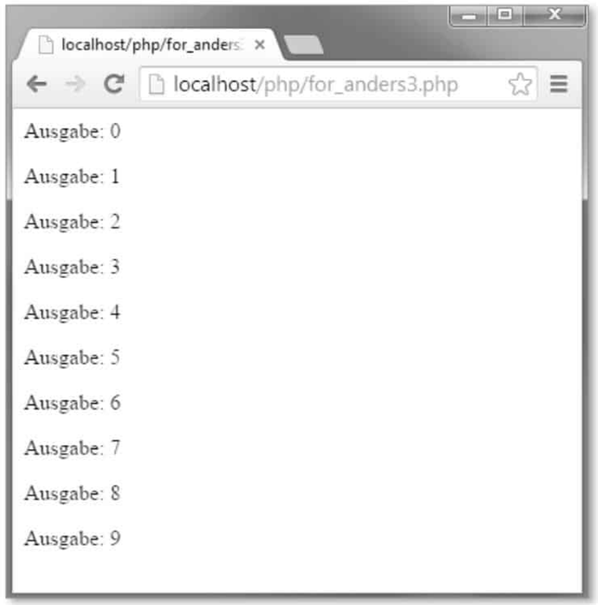


Abbildung 5.19 Der HTML-Code wird ausgegeben.

Schleifen verschachteln

Schleifen lassen sich in PHP beliebig verschachteln. Sie schreiben einfach die eine Schleife in den Anweisungsblock der anderen Schleife:

```
for ($i = 1; $i <= 10; $i++) {
  echo "Reihe $i: ";
  for ($j = 1; $j <= 10; $j++) {
    echo $j * $i . " ";
  }
  echo "<br />";
}
```

Listing 5.21 Verschachtelte »for«-Schleifen (»for_verschachtelt.php«)

Dieses Beispiel bildet zehn Reihen, die jeweils die Zahlen von 1 bis 10 multipliziert mit der Reihennummer enthalten (siehe Abbildung 5.20).

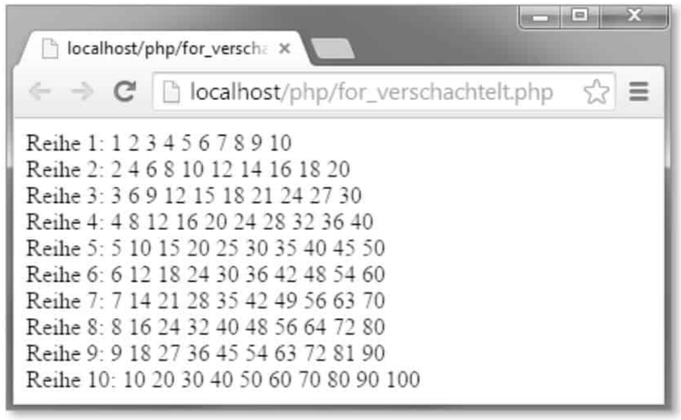


Abbildung 5.20 Das kleine Einmaleins

In der Praxis kommen verschachtelte Schleifen beispielsweise zum Einsatz, wenn Sie mit PHP eine Grafik manipulieren möchten. Um jedes Pixel eines Bilds umzufärben, verwenden Sie eine `for`-Schleife, die alle horizontalen Pixelspalten durchgeht, und darin verschachtelt eine `for`-Schleife, die sich alle Pixelreihen vornimmt. Mit diesen zwei verschachtelten `for`-Schleifen erwischen Sie jedes Pixel des Bilds. Ein weiteres Einsatzgebiet sind multidimensionale Arrays, die Sie in Kapitel 8, »Arrays«, kennenlernen werden.

5.3.2 »while«

Die `while`-Schleife gilt als Mutter der Schleifen. Mutter deswegen, weil sie in den meisten Programmiersprachen vorkommt und aus ihr die anderen Schleifenarten gebildet werden können.

Die `while`-Schleife besitzt nur eine »eingebaute« Funktionalität, nämlich die Bedingung:

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

Die Schleife wird so lange durchlaufen, wie die Bedingung wahr ist. Damit die Schleife allerdings irgendwann abbricht, muss sich ein Teil der Bedingung ändern. Die `for`-Schleife bietet für diesen Zweck die Durchlaufanweisung, in der `while`-Schleife müssen Sie sie selbst basteln.

Sehen Sie sich hierzu das folgende Beispiel an:

```
<?php
  $i = 1;
  while ($i < 10) {
    echo "$i<br />";
    $i++;
  }
?>
```

Listing 5.22 Die »while«-Schleife (»while.php«)

Die Variable `$i` ist die Zählervariable. Sie wird vor der `while`-Schleife initialisiert. Die `while`-Schleife selbst überprüft nur, ob `$i` kleiner als `10` ist. Solange dies der Fall ist, werden die Anweisungen ausgeführt. In dem Anweisungsblock befindet sich auch die Durchlaufanweisung. `$i` wird in jedem Schleifendurchlauf um `1` erhöht.

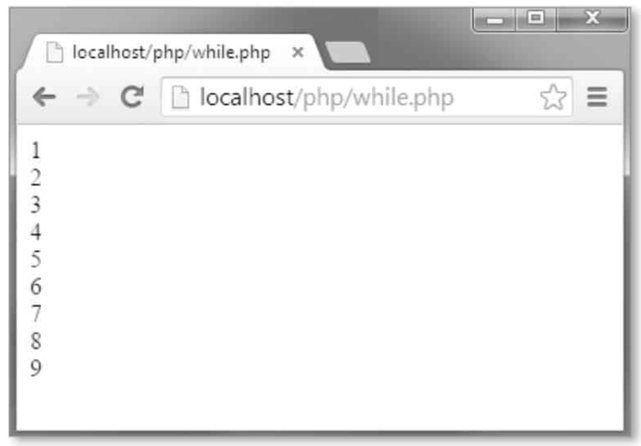


Abbildung 5.21 Die Zahlen von 1 bis 9 mit der »while«-Schleife

Hinweis

Wollten Sie eine Parallele zur `for`-Schleife ziehen, sähe das syntaktisch so aus:

```
Startanweisung;
while (Bedingung) {
  Anweisungen;
  Durchlaufanweisung;
}
```

Andere Formen

Auch die `while`-Schleife erlaubt einige andere Schreibweisen. Wer gern verkürzt, hat folgende Möglichkeiten:

1. Wenn nur eine Anweisung in der `while`-Schleife steht, können Sie die geschweiften Klammern weglassen und sogar alles in eine Zeile schreiben. Das ist zwar wie bei der `for`-Schleife, aber wo kommt dann die Durchlaufanweisung hin? Die können Sie (natürlich nur in einfachen Fällen) per Inkrement oder Dekrement direkt in die eine Anweisung einfügen. Im folgenden Fall wird das Inkrement nach dem Operanden geschrieben. Dadurch wird es auch erst nach der Anweisung ausgeführt.

```
$i = 1;
while($i < 10) echo $i++ . "<br />";
```

Listing 5.23 Dieses Listing gibt die Zahlen von 1 bis 9 aus (»while_anders.php«).

2. Auch für die while-Schleife gibt es die Doppelpunkt-Syntax, die hier mit `endwhile` beendet wird. Damit ist die `while`-Schleife ähnlich wie mit geschweiften Klammern flexibel über mehrere PHP-Blöcke hinweg einsetzbar:

```
<?php
$i = 1;
while ($i < 10):
?>

<p>Ausgabe: <?=$i ?></p>

<?php
$i++;
endwhile;
?>
```

Listing 5.24 »while« über mehrere PHP-Blöcke verteilt (»while_anders2.php«)

»break« und »continue«

Die Anweisung `break` dient dazu, eine Schleife zu verlassen. Möglich wäre also folgendes Konstrukt:

```
<?php
  $i = 1;
  while (true) {
    if ($i < 10) {
      echo "$i<br />";
      $i++;
    } else {
      break;
    }
  }
?>
```

Listing 5.25 Die Schleife wird ausschließlich mit »break« beendet (»break.php«).

Hierbei handelt es sich um eine provozierte Endlosschleife, die mit `break` beendet wird. Sie gibt die Zahlen von 1 bis 9 aus. Sie könnten `break` alternativ mit einer Zahl schreiben, die angibt, wie viele Schleifen von innen nach außen verlassen werden:

```
break 1;
```

entspricht

```
break;
```

Wenn Sie nun aber mehrere Schleifen oder zusätzlich eine `switch-case`-Anweisung verschachteln, können Sie auch höhere Werte angeben, um die Verschachtelung zu verlassen. Das folgende Beispiel zeigt eine `while`-Schleife, die eigentlich die Zahlen von 1 bis 9 ausgeben würde. Per `switch`-Anweisung werden zusätzlich zwei Fälle für das Produkt der Zahl mit `2` überprüft. Ist das Produkt `10`, wird nur die `switch`-Anweisung verlassen (und erst beim nächsten Schleifendurchlauf wieder geprüft). Ist das Produkt dagegen `16`, verlässt die `break`-Anweisung die `switch`-Fallunterscheidung (Nummer 1) und die `while`-Schleife (Nummer 2). Das heißt, die `9` wird nicht mehr ausgegeben.

```
<?php
  $i = 1;
  $j = 2;
  while ($i < 10) {
    echo "$i";
    switch ($i * $j) {
      case 10:
      echo " * $j = 10";
      break;
      case 16:
      echo " * $j = 16";
      break 2;
    }
    echo "<br />";
    $i++;
  }
?>
```

Listing 5.26 »break« mit numerischer Angabe (»break_nummer.php«)

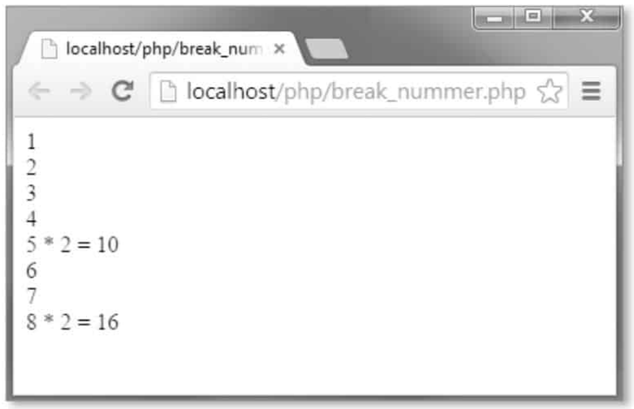


Abbildung 5.22 Bei 8 wird die Schleife abgebrochen.

Etwas seltener als `break` kommt `continue` zum Einsatz. `continue` bricht den Schleifendurchlauf ab, macht dann aber mit dem nächsten Durchlauf weiter. Das folgende Beispiel illustriert dies. Hier werden die Zahlen von 1 bis 9 ausgegeben. Wenn eine Zahl durch zwei teilbar ist[8], wird die Durchlaufanweisung erhöht und dann mit `continue` zum nächsten Schleifendurchlauf gewechselt. Die Anweisungen danach werden ignoriert. Für ungerade Zahlen ist die Bedingung dagegen nicht erfüllt, der Part mit `continue` wird ignoriert, und Durchlaufanweisung und Ausgabe `ungerade Zahl` werden ausgeführt:

```
<?php
  $i = 1;
  while ($i < 10) {
    echo "<br />$i";
    if ($i % 2 == 0) {
      $i++;
      continue;
    }
    $i++;
    echo " ungerade Zahl";
  }
?>
```

Listing 5.27 »continue« im Einsatz (»continue.php«)

8 Die Teilbarkeit durch 2 ist gegeben, wenn der Modulo, also der ganzzahlige Rest der Division, durch 2 gleich 0 ist. Diese Überprüfung wird recht häufig eingesetzt. Vorsicht, dieser einfache Test würde auch 0 als gerade Zahl erkennen! Alternativ können Sie übrigens eine Überprüfung mit bitweisem Operator einsetzen. `$i & 0` prüft z. B. auch, ob eine Zahl gerade ist.

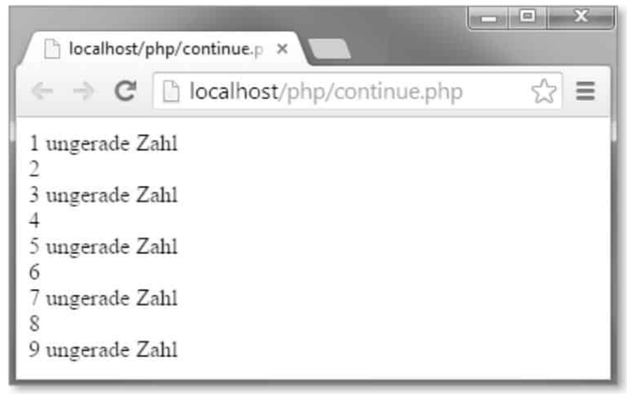


Abbildung 5.23 Die Ausgabe »ungerade Zahl« erfolgt in diesem Beispiel nur, wenn nicht vorher »continue« zum Einsatz kam.

`continue` kann wie `break` auch einen numerischen Wert erhalten, der bei verschachtelten Schleifen oder `switch`-Anweisungen verrät, bei welcher weitergemacht werden soll. Das folgende Beispiel zeigt dies anhand zweier verschachtelter Schleifen:

```
<?php
for ($i = 1; $i <= 10; $i++) {
  echo "Reihe $i: ";
  $j = 1;
  while (true) {
    echo $j * $i . " ";
    $j++;
    if ($j > 5) {
      echo "<br />";
      continue 2;
    }
  }
}
?>
```

Listing 5.28 »continue« mit numerischer Angabe (»continue_nummer.php«)

`break` und `continue` funktionieren auch mit den anderen Schleifenarten genau wie hier erläutert. Am häufigsten kommen sie in der Praxis mit `while` zum Einsatz.

Hinweis

Bis PHP 5.3 gab es noch ein Konstrukt, in dem `break` und `continue` auch mit einer Variablen für die numerische Angabe verbunden werden konnte. Diese Möglichkeit wurde in der Praxis fast nie verwendet und in PHP 5.4 entfernt.

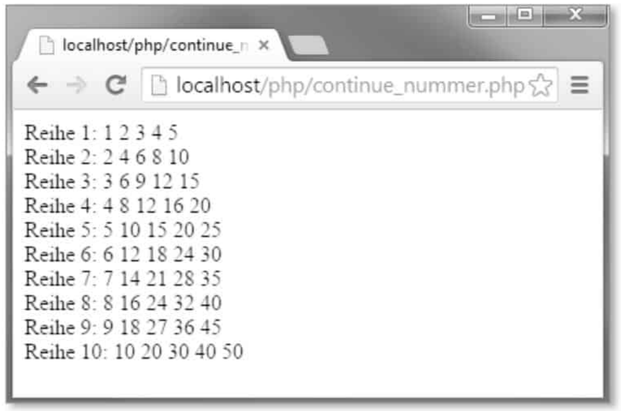


Abbildung 5.24 Die Reihen des kleinen Einmaleins bis zur Multiplikation mit 5

5.3.3 »do-while«

Die letzte Schleifenart, die hier vorgestellt werden soll, ist `do-while`. Sie funktioniert im Prinzip wie die `while`-Schleife mit der einzigen Ausnahme, dass die Bedingung immer erst nach dem Schleifendurchlauf geprüft wird. Das heißt, dass die Anweisungen mindestens einmal ausgeführt werden.

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

Das folgende Beispiel zeigt die gewohnte Schleife, die die Zahlen von 1 bis 9 ausgibt.

```
<?php
  $i = 1;
  do {
    echo "$i<br />";
    $i++;
  } while ($i < 10)
?>
```

Listing 5.29 Die »do-while«-Schleife (»dowhile.php«)

Soweit also keine beobachtbare Abweichung zur normalen `while`-Schleife. Ungewöhnlich wird es erst, wenn die Bedingung von Anfang an nicht erfüllt ist. In diesem Fall wird nämlich der Anweisungsblock zumindest einmal ausgeführt:

```
$i = 11;
do {
  echo "$i<br />";
  $i++;
} while ($i < 10)
```

Listing 5.30 Die Besonderheit von »do-while« (»dowhile_einmal.php«)

Im Beispiel gibt `do-while` zumindest einmal `11` aus, obwohl die Bedingung nicht erfüllt ist. Dieses Verhalten wird zwar selten benötigt, wenn Sie es aber dennoch mal brauchen, erinnern Sie sich an die `do-while`-Schleife.



Abbildung 5.25 Obwohl die Bedingung nicht erfüllt ist, gibt »do-while« einmal »11« aus.

Hinweis

`do-while` besitzt keine Kurzformen. `break` und `continue` können Sie allerdings mit `do-while` einsetzen.

5.4 Sprünge

Seit PHP 5.3 gibt es eine Neuerung, die durchaus umstritten war: Es wurden Sprünge im Codeablauf möglich. Eingeführt wurde der Operator `goto`. Er lässt sich sehr einfach einsetzen: Sie fügen eine Marke ein. Der Name ist frei wählbar und wird mit einem Doppelpunkt ergänzt. Mit `goto Markenname` rufen Sie dann die Marke auf. Sämtlicher Code dazwischen wird übersprungen. Im folgenden Beispiel wird `Ausgabe 1` ignoriert und dann `Ausgabe 2` ausgegeben:

```
<?php
goto Marke;
echo 'Ausgabe 1';

Marke:
echo 'Ausgabe 2';
?>
```

Listing 5.31 Ein Sprung per »goto« (»goto.php«)



Abbildung 5.26 Nur die zweite Ausgabe ist zu sehen.

Hinweis

`goto` erlaubt keinen Sprung in eine Schleife oder `switch`-Fallunterscheidung. Dies würde einen *Fatal error* erzeugen.

Kapitel 7
Strings

String-Behandlung – mancher bricht da schon in Gähnen aus. Leider gehören Strings zu den wichtigsten Werkzeugen eines Programmierers. Dieses Kapitel zeigt Ihnen, was Sie in der Praxis benötigen.

Ein großer Prozentsatz aller Daten landet als String bei Ihnen, egal, ob als Rückgabewert einer Funktion oder als Eingabe des Benutzers. Ein Blick in die Onlinedokumentation von PHP raubt einem zuerst mal den Atem. Die Liste an Funktionen zur String-Manipulation ist ellenlang (*http://www.php.net/manual/de/ref.strings.php*). Aber keine Sorge, hier finden Sie Beispiele und Erklärungen für die wichtigen String-Manipulationen.

Hinweis

Wenn Sie noch mehr Power benötigen, sollten Sie die regulären Ausdrücke genauer unter die Lupe nehmen. PHP übernimmt hier die Funktionalität von Perl, mehr dazu in Kapitel 10, »Reguläre Ausdrücke«. Einige der String-Funktionalitäten erfordern außerdem Wissen über Arrays, die Sie in Kapitel 8, »Arrays«, finden. Wir haben dennoch Strings vorangestellt, da Arrays logisch darauf aufbauen und auch bei Arrays die String-Behandlung eine Rolle spielt.

7.1 Verbinden

Das Verbinden von Strings heißt im Programmierchinesisch auch *Konkatenation*. Strings werden also konkateniert. Dies geschieht in PHP mit dem Punkt (`.`). Dies haben Sie schon viele Male verwendet und/oder in Kapitel 5, »Programmieren«, gelesen. Hier interessieren allerdings noch die Details: Was passiert eigentlich, wenn ein String mit anderen Datentypen verbunden wird? In PHP ist das überhaupt kein Problem, denn da der Punkt speziell für String-Operationen reserviert ist, wandelt PHP immer vorher die Datentypen in Strings um. Das heißt, dieser Code:

```
$a = 20;
$b = " Euro";
echo $a . $b;
```

gibt korrekterweise `20 Euro` aus. Der folgende Code gibt dagegen was aus?

```
$a = 20;
$b = 40;
echo $a . $b;
```

Richtig, beide Integer werden in einen String umgewandelt. Das Ergebnis ist also 2040.

7.2 Teilen und Zusammenfügen

Eine der häufigsten Aufgaben besteht darin, einen String in seine Einzelteile zu zerlegen oder aus Einzelteilen wieder einen String zu machen. Allein dafür bietet PHP schon viele verschiedene Funktionen.

7.2.1 Auf Zeichen zugreifen

Sie können in PHP mit geschweiften oder eckigen Klammern auf einzelne Buchstaben eines Strings zugreifen wie auf die Elemente eines Arrays. Der erste Buchstabe hat den Index 0:

```
$text = "Beispielstring";
echo $text[0];
```

liefert also als Ergebnis das B.

> **Hinweis**
>
> Lange Zeit galt die Variante mit eckigen Klammern als nicht mehr empfohlen. Im Zuge der turnusmäßigen Bereinigungen für PHP entschied man sich dann allerdings dazu, die ungewöhnlichere Syntax mit geschweiften Klammern zu entfernen. In PHP 5.1 liefern geschweifte Klammern dementsprechend schon beim Fehlerlevel E_STRICT einen Fehler, später verschwinden sie ganz.

7.2.2 Gleichmäßig unterbrechen

Die Funktion chunk_split(String, Größe, Einfügen) teilt einen String in gleich große Teile. Zwischen diese Teile können Sie eine andere Zeichenkette einfügen. Die Funktion liefert dann als Rückgabe das Ergebnis. Im folgenden Beispiel trennen wir die Zeichenkette nach jeweils vier Zeichen. Nach jeder Trennung fügen wir eine horizontale Linie mit dem HTML-Tag <hr /> ein:

```
<?php
  $a = "PHP ist toll.";
```



```
  echo chunk_split($a, 4, "<hr />");
?>
```

Listing 7.1 »chunk_split()« (»chunk_split.php«)

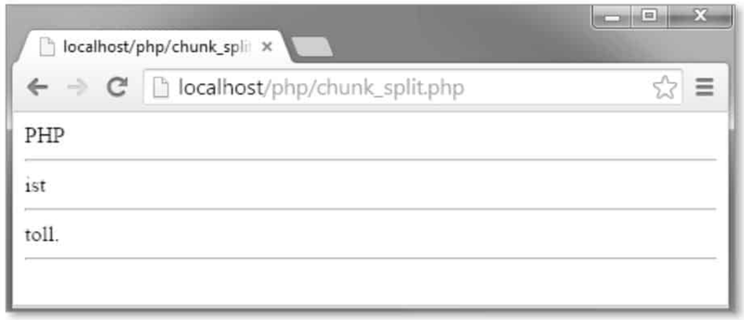


Abbildung 7.1 Der Füllstring wird nach jedem Vorkommen eingefügt.

Einziges Problem bei dieser Funktion ist, dass der Füllstring, wie in Abbildung 7.1 zu sehen, nach jeder Teilung eingefügt wird, also auch am Ende. Wenn Sie das nicht möchten, müssen Sie die letzten Zeichen wegschneiden. Dies erledigen Sie mit der Funktion substr(String, Startposition, Zeichenzahl). Sie geben als Startposition zum Abschneiden einfach 0, also den Anfang des Strings, an. Als Zeichenzahl wählen Sie einen negativen Wert. Damit Sie diesen nicht immer manuell an die Länge Ihres Trennstrings anpassen müssen, bestimmen Sie dessen Länge mit der Funktion strlen(String):

```
<?php
  $a = "PHP ist toll";
  $trenn = "<hr />";
  echo substr(chunk_split($a, 4, $trenn), 0, -strlen($trenn));
?>
```

Listing 7.2 »chunk_split()« mit abgeschnittenem Trennzeichen (»chunk_split2.php«)

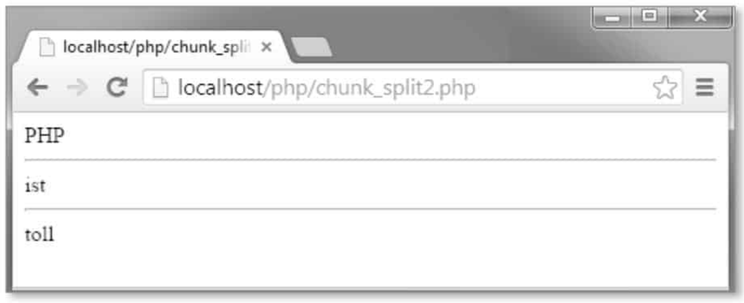


Abbildung 7.2 Keine Linie mehr am Ende

7.2.3 Zeilenumbrüche

Die Methode `wordwrap(String, Länge, Trennzeichen, Abschneiden)` arbeitet ähnlich wie `chunk_split()`. Ihr eigentliches Ziel ist allerdings, Zeilenumbrüche nach Wörtern einzufügen. Dabei sind alle Parameter bis auf den `String` selbst optional. Die `Länge` gibt an, nach wie vielen Zeichen umbrochen wird. Wenn Sie sie weglassen, wird automatisch nach 75 Zeichen umbrochen. Das `Trennzeichen` ist ein String, der an der Stelle des Umbruchs eingefügt wird. Lassen Sie ihn weg, fügt PHP einen Zeilenumbruch mit `\n` ein. Der Parameter `Abschneiden` bestimmt, ob einzelne Wörter durchgeschnitten werden (Wert `1` für `true`). Standardmäßig ist dies deaktiviert und hat den Wert `0`.

Hinweis

Wenn Sie `Abschneiden` aktivieren, funktioniert `wordwrap()` genauso wie `chunk_split()`.

Im folgenden Skript werden im String jeweils nach drei Zeichen Umbrüche eingefügt. Da allerdings ganze Wörter erhalten bleiben, entsteht die Trennung nur bei den einzelnen Wörtern `PHP`, `ist` und `toll`.

```
<?php
  $a = "PHP ist toll";
  echo wordwrap($a, 3);
?>
```

Listing 7.3 »wordwrap()« (»wordwrap.php«)

Im Browser sehen Sie bei diesem Beispiel gar nichts. Warum? Der Umbruch erfolgt mit `\n`, zu sehen ist er nur im Quelltext. HTML kennt `\n` dagegen nicht und ignoriert es.



Abbildung 7.3 Der Umbruch ist nur im Quelltext zu sehen.

Sie könnten nun als Trennzeichen einfach `<br />` angeben:

```
wordwrap($a, 3, "<br />");
```

Oder Sie verwenden die Funktion `nl2br(String)`. Sie wandelt in einem String alle Umbrüche mit `\n` in Umbrüche mit dem HTML-Tag `<br />` um:


```
<?php
  $a = "PHP ist toll";
  echo nl2br(wordwrap($a, 3));
?>
```

Listing 7.4 »nl2br()« (»nl2br.php«)

Übrigens, `nl2br()` erhält die Umbrüche mit `\n` im Quellcode bzw. im String.

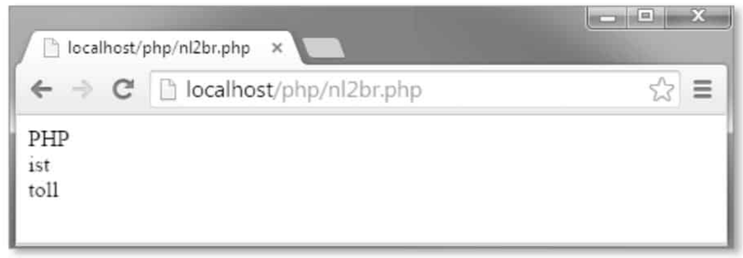


Abbildung 7.4 Nun klappt der Umbruch auch im Browser.

Hinweis

`wordwrap()` gibt es seit PHP 4.0.2, den optionalen vierten Parameter erst seit PHP 4.0.3. `nl2br()` kennt erst ab PHP 4.0.5 den Zeilenumbruch in XHTML-Schreibweise mit `<br />`. Diese oder ältere Versionen sind allerdings schon allein wegen der Sicherheitsbugs heute kaum noch im Einsatz. Eng verwandt mit `wordwrap()` ist `str_word_count(String)`, das die Anzahl der Wörter in einem String zählt. Diese Funktion kann allerdings auch die Wörter in ein Array teilen.

7.2.4 Teilen in Strings

Vorgefertigte Funktionen helfen dann nicht mehr weiter, wenn Sie kein Trennzeichen einfügen, sondern aus einem String Einzelstrings machen möchten.[1]

Gleichmäßig teilen

Um Strings gleichmäßig in Strings zu teilen, verwenden Sie am einfachsten `substr()` und `strlen()` in Verbindung mit variablen Variablen.[2]

1 Dies kommt in der Praxis ab und an vor. Oftmals wollen Sie aber einen String in Einzelteile zerlegt in ein Array speichern. Davon handelt der Abschnitt 7.2.5, »Strings und Arrays«. Dort werden Trennzeichen verwendet, um die Einzelteile zu identifizieren. Sie können auch die dort gezeigten Funktionen mit `chunk_str()` verbinden. Sie sehen, String-Behandlung ist ein komplexes Feld und gar nicht so langweilig, wie man denken könnte.

2 Siehe dazu den entsprechenden Abschnitt in Kapitel 4, »Grundlagen der Sprache«.

Und so geht es: Die Schleife beinhaltet zwei Zählervariablen. `$i` für die Schritte, für die jeweilige Startposition, und `$j` für den variablen Variablennamen – ein schönes Wort. `strlen()` stellt die Länge des Strings fest und bestimmt damit, wann die letzte Startposition erreicht ist. Die variable Variable wird aus einem String und `$j` gebildet, da eine Zahl allein nicht als variable Variable eingesetzt werden kann. Hier das Skript:

```
<?php
  $a = "PHP ist toll";
  for ($i = 0, $j = 0; $i < strlen($a); $i = $i + 4, $j++) {
    $name = "string" . $j;
    $$name = substr($a, $i, 4);
  }
?>
```

Listing 7.5 Teilen mit »substr()« und einer Schleife (»teilen_schleife.php«)

Mit Trennzeichen teilen

Für das Teilen mit Trennzeichen in Einzelstrings[3] bietet PHP die Funktion `strtok(String, Trennzeichen)`. Sie liefert als Rückgabe den ersten Teil eines Strings bis zum Trennzeichen. Wenn Sie beim zweiten Aufruf der Funktion den String selbst weglassen und nur das Trennzeichen angeben, erhalten Sie den zweiten Teil des vorher angegebenen Strings. Hier ein Beispiel:

```
<?php
  $a = "PHP ist toll";
  echo "Teil 1: " . strtok($a, " ") . "<br />";
  echo "Teil 2: " . strtok(" ") . "<br />";
  echo "Teil 3: " . strtok(" ");
?>
```

Listing 7.6 »strtok()« (»strtok.php«)

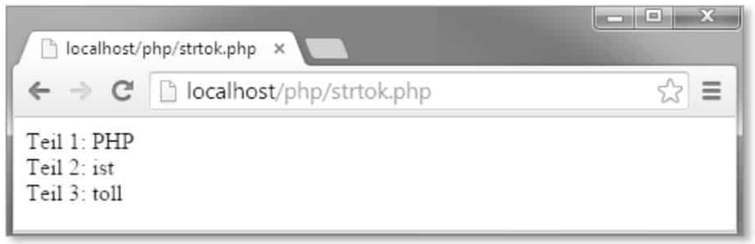


Abbildung 7.5 Drei Aufrufe von »strtok()« liefern drei Teile.

3 Vorsicht, Verwechslungsgefahr! Für das Teilen eines Strings in ein Array – in der Realität oft praktikabler – gibt es andere Funktionen, die Sie im Abschnitt 7.2.5, »Strings und Arrays«, kennenlernen.

Das Ganze klappt natürlich auch mit einer Schleife:

```
$a = "PHP ist toll";
$i = 1;
$start = strtok($a, " ");
while ($start) {
  $name = "teil" . $i;
  $$name = $start;
  $start = strtok(" ");
  $i++;
}
```

Listing 7.7 »strtok()« mit Schleife (»strtok_schleife.php«)

7.2.5 Strings und Arrays

Zum Teilen und Zusammenfügen von Strings gehört auch die Umwandlung von Strings in ein Array und umgekehrt. Für beides gibt es unterschiedlich mächtige Funktionen.

String zu Array

Beginnen wir einfach: Die Funktion `explode(Trennzeichen, String, Limit)` teilt einen String an bestimmten Trennzeichen in ein Array. Wird der optionale Parameter `Limit` angegeben, werden nur so viele Array-Elemente erzeugt, wie das `Limit` vorgibt. Das letzte Array-Element enthält den Rest des Strings unabhängig von seiner Länge.

Hier ein einfaches Beispiel: Zuerst teilt das Skript den String bei jedem Leerzeichen und gibt dann die einzelnen Elemente des Arrays aus:

```
<?php
  $a = "PHP ist toll";
  $strings = explode(" ", $a);
  foreach ($strings as $element) {
    echo $element . "<hr />";
  }
?>
```

Listing 7.8 »explode()« (»explode.php«)

Was aber, wenn Sie beispielsweise mehrere Trennzeichen berücksichtigen möchten? In diesem Fall gibt es einige andere Ansätze, die die folgenden Abschnitte näher erläutern.

Eigene Funktionen | Gerade für das Trennen schreibt man sich oft am besten schnell eine eigene Funktion. Sie finden davon einige in der PHP-Dokumentation vor allem in den nützlichen Kommentaren der User.

Hier ein eigenes Beispiel, das wir auch in der Praxis eingesetzt haben. Die folgende Funktion übernimmt ein Array mit Trennzeichen und trennt anhand dieser einen String. Alle Trennzeichen werden dann mittels `str_replace(Zu Ersetzen, Ersatz, String)` mit dem ersten Trennzeichen des Arrays ersetzt. Dann wird der String anhand dieses Trennzeichens mit `explode()` getrennt. Nun gibt es nur noch ein Problem: Wenn zwei Trennzeichen in dem String hintereinander folgen, entsteht ein leeres Array-Element. Alle leeren Elemente filtern wir mit der Funktion `array_filter(Array, Funktion)` heraus. Liefert die Funktion `false`, wird das Element im von der Funktion zurückgegebenen Array nicht berücksichtigt:

```
function explode_mehrere($trennzeichen, $string) {
  $string = str_replace($trennzeichen, $trennzeichen[0], $string);
  $ergebnis = explode($trennzeichen[0], $string);
  $ergebnis = array_filter($ergebnis, "filtern");
  return $ergebnis;
}
function filtern($wert) {
  if($wert == "") {
    return false;
  } else {
    return true;
  }
}
```

Listing 7.9 Mehrere Trennzeichen (»explode_mehrere.php«)

Verwenden Sie beispielsweise den folgenden String mit der Funktion:

```
$a = "PHP ist toll. Und alles ist gut.";
```

und geben Sie das zurückgelieferte Array aus:

```
$strings = explode_mehrere(array(" ", "."), $a);
foreach ($strings as $element) {
  echo $element . "<hr />";
}
```

Das Ergebnis sehen Sie in Abbildung 7.6.

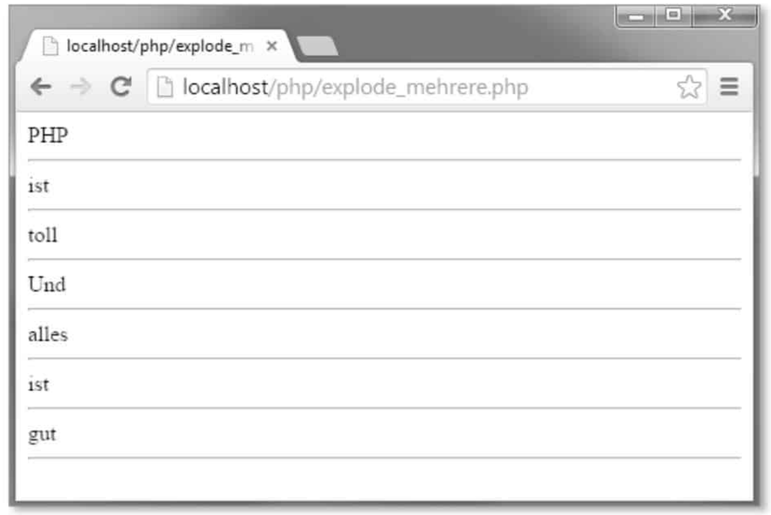


Abbildung 7.6 Der String wurde anhand von Leerzeichen und Punkt in Einzelteile zerlegt.

Tipp

Kein spektakulärer Tipp, aber eine Binsenweisheit, an die Sie ab und an denken sollten: Oftmals führt nur die Kombination mehrerer Mittel – und im PHP-Fall meist Funktionen – zum Erfolg. Wenn Sie ein bestimmtes Problem lösen wollen, schauen Sie zuerst einmal, ob es nicht so ähnliche Funktionen gibt, bevor Sie sich daranmachen, sie selbst komplett zu schreiben.

»split()« und »preg_split()« – reguläre Ausdrücke | Die Funktionen `split()` und `preg_split()` sind Alternativen zu `explode()`. Sie verwenden einen regulären Ausdruck als Suchmuster für die Trennung. Mehr dazu erfahren Sie in Kapitel 10, »Reguläre Ausdrücke«.

Hinweis

`explode()` ist etwas performanter als die Alternativen mit regulären Ausdrücken, da die regulären Ausdrücke erst noch interpretiert werden müssen. Wenn Sie eigene Funktionen schreiben, sollten Sie testen, welche Lösung schneller ist. Probieren Sie es einfach mal mit einem sehr langen String aus.

»str_split()« ab PHP 5 | `str_split(String, Länge)` gibt es ab PHP 5.0. Mit dieser Funktion teilen Sie einen `String` in Stücke gleicher `Länge` und speichern die Stücke in ein Array. Wenn Sie die `Länge` weglassen, ist jedes Zeichen ein Array-Element. Letzteres sehen Sie im folgenden Beispiel:

```
<?php
  $a = "PHP ist toll";
  print_r (str_split($a));
?>
```

Listing 7.10 »str_split()« (»str_split.php«)

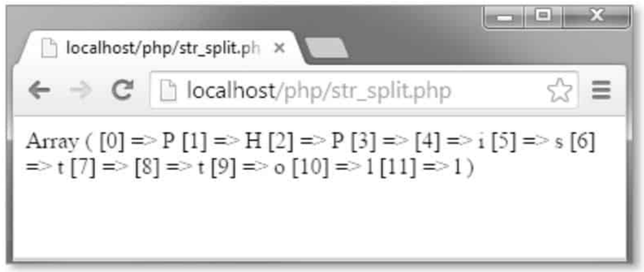


Abbildung 7.7 Das Array zeigt auch Leerzeichen als Elemente.

Für PHP 4 können Sie sich diese Funktion allerdings auch selbst schreiben. Nehmen Sie als Basis einfach das Skript *teilen_schleife.php*. In der Schleife benötigen Sie eine Variable als Zähler für den Index des Arrays (`$j`) und eine für die jeweilige Startposition von `substr()`. Letztere ist auch gleichzeitig in der Schleifenbedingung relevant, die abbricht, wenn das Ende des Strings erreicht ist.

```
function str_split_php4($string, $laenge = 1) {
  $ergebnis = array();
  for ($i = 0, $j = 0; $i < strlen($string); $i += $laenge, $j++) {
    $ergebnis[$j] = substr($string, $i, $laenge);
  }
  return $ergebnis;
}
```

Listing 7.11 »str_split()« für PHP 4 simulieren (»str_split_simulieren.php«)

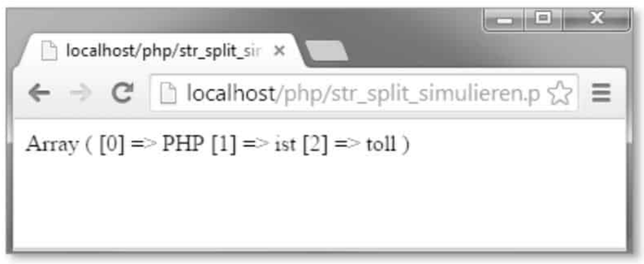


Abbildung 7.8 Die Simulation hat geklappt.

»str_word_count()« | Wenn `str_word_count(String)` nur einen String als Parameter erhält, liefert sie einen Integer mit der Zahl der Wörter eines Strings. Geben Sie aber zusätzlich `str_word_count(String, Format)` an, liefert die Funktion entweder ein normales Array mit allen Wörtern (`Format` hat den Wert `1`) oder ein assoziatives Array mit der Position des Worts als Schlüssel und dem Wort als Wert (`Format` mit Wert `2`).

> **Hinweis**
>
> `str_word_count()` gibt es seit PHP 4.3.0. Als dritten Parameter können Sie seit PHP 5.1 eine Liste mit Zeichen angeben, die als Trenner eines Worts akzeptiert werden.


```
<?php
  $a = "Always look on the bright side of life!";
  print_r(str_word_count($a, 1));
?>
```

Listing 7.12 Aus dem String wird ein Array (»str_word_count_array.php«).

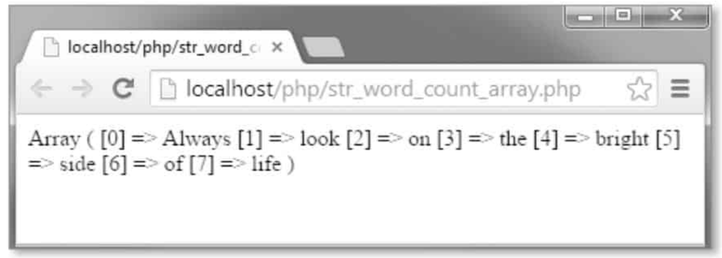


Abbildung 7.9 Das Array mit den einzelnen Wörtern ohne Leerzeichen

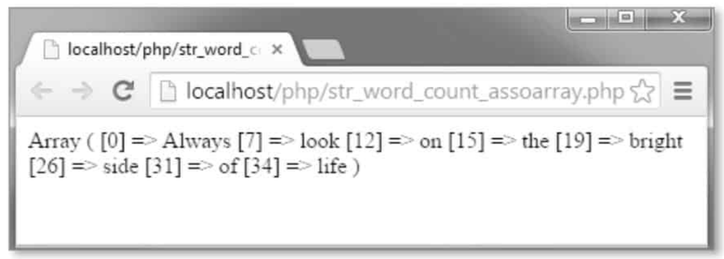


Abbildung 7.10 Und hier mit assoziativem Array, das die Startposition der einzelnen Wörter zeigt (erreicht durch »Format« mit Wert »2«)

> **Hinweis**
>
> Vorsicht, einen Standardwert für `Format` gibt es nicht! Nur wenn der Parameter weggelassen wird, liefert die Funktion die Zahl der Wörter als Integer.

Array zu String

Um ein Array in einen String zu verwandeln, verwenden Sie die Methode `implode(Verbindungszeichen, Array)`. Wenn Sie das Verbindungszeichen weglassen, werden die Elemente einfach direkt aneinandergehängt.

> **Hinweis**
>
> Sie können auch `implode(Array, Verbindungszeichen)` schreiben, also die Parameter vertauschen. Dies ist allerdings eher ungebräuchlich, da `explode()` das Trennzeichen immer voranstellt. Da es nicht einmal offiziell dokumentiert ist, könnte die Funktionalität auch bald verschwinden – Sie sollten sich also nicht darauf verlassen.

```
<?php
  $werte = array("PHP", "ist", "toll");
  $ergebnis = implode(" ", $werte);
  echo $ergebnis;
?>
```

Listing 7.13 »implode()« (»implode.php«)

Das Ganze funktioniert natürlich auch mit einem assoziativen Array.[4] Hier werden die Elemente in der Reihenfolge der Definition aneinandergehängt:

```
<?php
  $werte = array("R" => "FF", "G" => "AA", "B" => "00");
  $ergebnis = implode($werte);
  echo "Farbwert: #" . $ergebnis;
?>
```

Listing 7.14 »implode()« mit assoziativem Array (»implode_asso.php«)

Die Funktion `join(Verbindungszeichen, Array)` hat exakt die gleiche Wirkung wie `implode()`. `join()` wird deswegen auch als Alias von `implode()` bezeichnet.[5]

7.3 Groß- und Kleinschreibung

Im Web ist nur sehr schwer zu kontrollieren, wie ein Benutzer einen bestimmten Text in ein Formularfeld einträgt. Eines der wichtigsten Probleme ist die Unterscheidung von Groß- und Kleinschreibung.

4 Ein Array, das statt eines Index von 0 bis n Schlüsselwerte besitzt. Mehr dazu in Kapitel 8, »Arrays«.

5 Aliase haben meist historische Gründe: Eine Funktion ist unter einem Namen aus einer Programmiersprache bekannt und dann eben mit einem zweiten Namen implementiert. Eine Liste der Aliase in PHP finden Sie unter *http://www.php.net/manual/de/aliases.php*.

Zeichen, also Buchstaben, Ziffern und Sonderzeichen, lassen sich als ASCII-Code darstellen. Die ASCII-Codes der Buchstaben sind auch relevant für den String-Vergleich.[6] Großbuchstaben haben dort andere (niedrigere) ASCII-Codes als Kleinbuchstaben.

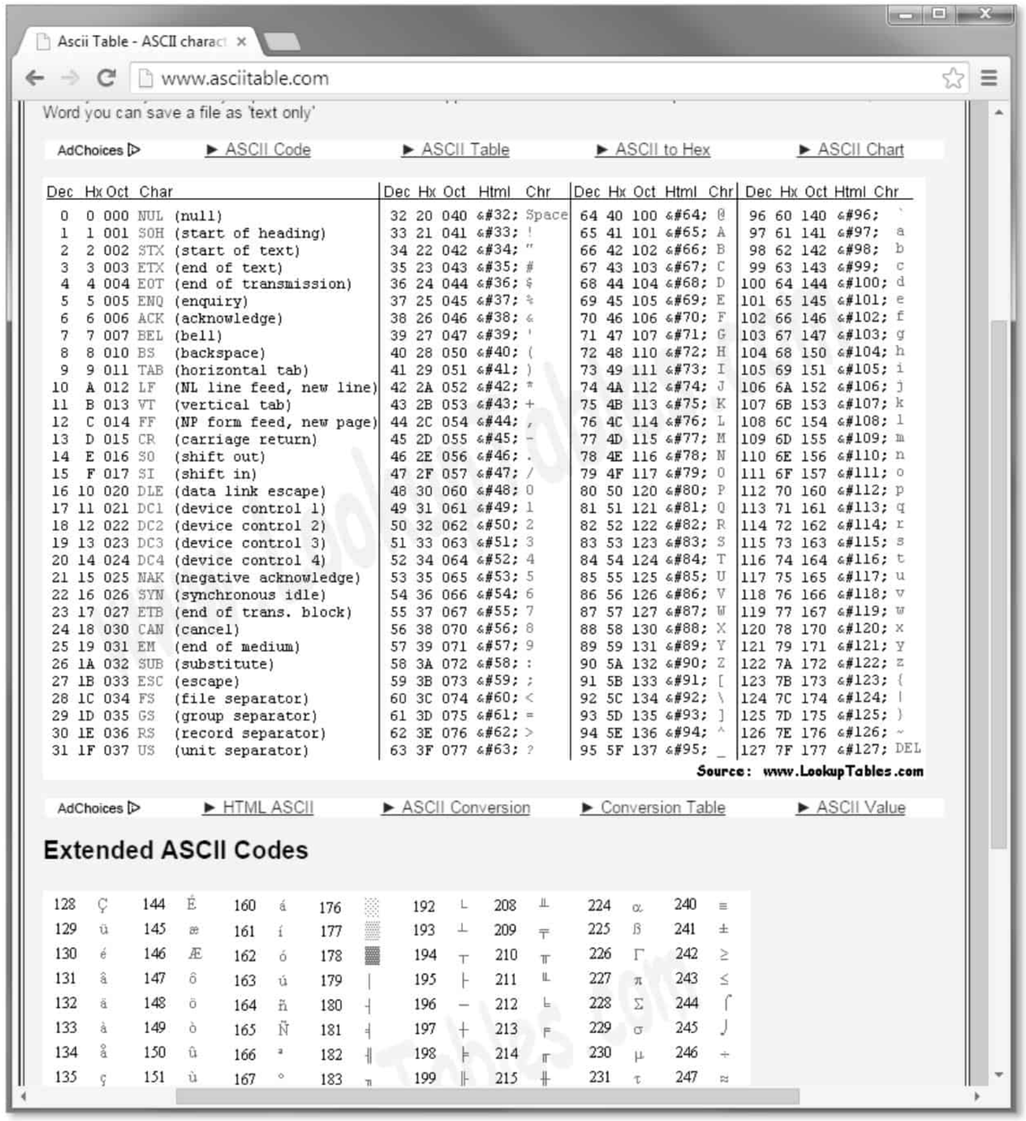


Abbildung 7.11 Eine bekannte ASCII-Tabelle finden Sie unter »http://www.asciitable.com«.

Um nun Groß- in Kleinbuchstaben umzuwandeln und umgekehrt, könnten Sie natürlich die ASCII-Codes filtern. Die PHP-Funktion `ord(String)` liefert den ASCII-Code eines Buchstabens, `char(Ascii)` ist das Gegenstück und macht aus einem ASCII-

6 Siehe Kapitel 5, »Programmieren«.

Code den zugehörigen String. Diese Arbeit wurde Ihnen allerdings von den PHP-Entwicklern schon abgenommen: `strtolower(String)` wandelt alle Buchstaben eines Strings in Kleinbuchstaben um, `strtoupper(String)` alle in Großbuchstaben. Andere Zeichen wie Ziffern oder Sonderzeichen bleiben unverändert.

```
<?php
  $a = "PHP ist toll";
  echo strtolower($a);
?>
```

Listing 7.15 »strtolower()« (»strtolower.php«)

Das obige Skript erzeugt beispielsweise folgende Ausgabe: `php ist toll`.

> **Tipp**
>
> Viele Funktionen, beispielsweise `str_replace()`, haben Varianten, die nicht zwischen Groß- und Kleinschreibung unterscheiden. Im Fall von `str_replace()` ist das `str_ireplace()`, das es allerdings erst in PHP 5 gibt. Bei den String-Vergleichsfunktionen erkennen Sie die case-insensitiven[7] Varianten an dem Begriff `case` im Namen.

Zwei Hilfsfunktionen fallen ebenfalls in den Bereich:

- `ucfirst(String)` verwandelt das erste Zeichen eines Strings in einen Großbuchstaben, wenn es denn vorher ein Kleinbuchstabe war.
- `ucwords()` macht alle Wortanfänge zu Großbuchstaben, aber natürlich auch nur, wenn es vorher Kleinbuchstaben waren.

Eine mögliche Praxisanwendung für die verschiedenen Funktionen rund um Groß- und Kleinschreibung ist ein Schreischutz für ein Forum. Das heißt, eine Funktion überprüft, ob viele Großbuchstaben in einem String vorkommen, die auf ein Schreien des Nutzers hindeuten. Wenn das der Fall ist, wird der String umgewandelt.

Die folgende einfache Funktion realisiert dies: Zuerst werden die einzelnen Wörter in ein Array aufgesplittet.[8] Bis auf den Anfangsbuchstaben wird dann jeder Buchstabe von jedem Wort durchlaufen. Dabei prüft eine Fallunterscheidung mittels der Funktion `ord()`, ob der ASCII-Code des jeweiligen Zeichens ein Klein- oder Großbuchstabe ist, und erhöht jeweils einen Zähler. Andere Zeichen werden ignoriert. Zum Schluss

7 Der (neudeutsche) Fachbegriff für »nicht zwischen Groß- und Kleinschreibung unterscheiden«. *Case* steht in diesem Kontext für Zeichen bzw. Buchstaben und kommt aus dem Sprachgebrauch des klassischen Schriftsatzes, als Buchdrucker ihre Druckplatten mit bleiernen Buchstaben aus einem Satzkasten gesetzt haben. Jeder Buchstabe hatte dabei ein Kästchen in diesem Kasten, deswegen case.

8 `str_word_count()` funktioniert erst ab PHP 4.3.0. Sie können alternativ auch `explode()`, eine eigene Funktion oder einen regulären Ausdruck einsetzen.

erfolgt ein Vergleich der Zähler. Den Wert, den Sie hier ansetzen, können Sie natürlich frei wählen. Wir wandeln den String in Kleinbuchstaben und die Wortanfänge in Großbuchstaben um, wenn mehr Groß- als Kleinbuchstaben vorhanden sind:

```
function schreischutz($string) {
  $worte = str_word_count($string, 1);
  $gross = 0;
  $klein = 0;
  foreach($worte as $wort) {
    for ($i = 1; $i < strlen($wort); $i++) {
      $ascii = ord(substr($wort, $i, 1));
      if ($ascii >= 65 && $ascii <= 90) {
        $gross++;
      } elseif ($ascii >= 97 && $ascii <= 122) {
        $klein++;
      }
    }
  }
  if ($gross > $klein) {
    return ucwords(strtolower($string));
  } else {
    return $string;
  }
}
```

Listing 7.16 Ein einfacher Schreischutz (»gross_klein.php«)

Gibt der Benutzer nun beispielsweise `PHP IST SCHLECHT` ein, wandelt das die Funktion in

```
Php Ist Schlecht
```

um. `PHP ist schlecht` wird dagegen so belassen, wie es ist. Sie könnten diese Funktion noch beliebig erweitern und verbessern. Beispielsweise ließen sich noch Umlaute überprüfen, oder Sie schützen einige Begriffe wie z. B. PHP vor der Umwandlung.

7.4 Beschneiden

Eine Funktion zum Ausschneiden von Teilen eines Strings haben Sie bereits kennengelernt: `substr()`. Sie findet in diesem Abschnitt ihre eigentliche Heimat. Dazu kommen einige weitere Hilfsfunktionen, beispielsweise um Leerzeichen zu entfernen.

7.4.1 Zeichen ausschneiden

`substr(String, Startposition, Länge)` besitzt drei Parameter:

- `String` gibt den String an, der zugeschnitten werden soll.
- `Startposition` regelt, wo das Abschneiden beginnt.
- `Länge` besagt, wie viele Zeichen abgeschnitten werden. Dieser Parameter ist optional. Wird er weggelassen, liefert `substr()` alle Zeichen ab der `Startposition` bis zum Ende des Strings aus.

Was `substr()` auszeichnet, ist eine relativ große Flexibilität. Sehen Sie sich dies an einigen Beispielen an. Ausgangspunkt ist der folgende String:

```
$a = "PHP ist toll";
```

Wenn Sie nur eine (positive) Startposition angeben, werden alle Zeichen bis zum Ende des Strings zurückgeliefert. Die folgenden Parameter

```
substr($a, 4)
```

liefern also `ist toll`.

```
substr($a, 4, 3)
```

ergibt dagegen nur drei Buchstaben, nämlich `ist`.

Und wie funktioniert es bei einer negativen Startposition? Hier wird von rechts gezählt. `-4` bedeutet also, der viertletzte Buchstabe des Strings ist die Startposition, und zwar unabhängig von der Länge des Strings.

```
substr($a, -4)
```

gibt also `toll` zurück.

Hinweis

Vorsicht, die erste Position in einem String ist die `0`! Wenn mit negativen Werten von hinten begonnen wird, hat das erste Zeichen von hinten dagegen die Position `-1`. Dies ist logisch, denn die Startposition des ersten Zeichens muss es ja auch geben, und das ist eben die `0`.

Von der Startposition wird immer nach rechts ausgeschnitten. Bei negativer `Länge` wird von hinten abgeschnitten:

```
substr($a, 1, -1)
```

schneidet also den ersten und den letzten Buchstaben ab und gibt `HP ist tol` aus.

Hinweis

`Länge` und `Startposition` lassen sich beliebig kombinieren, also beispielsweise auch negative Werte miteinander. Interessant wird es, wenn der String kürzer ist als die Angaben. Bei der `Länge` macht dies keine Schwierigkeiten. `substr()` liefert einfach nur so viele Zeichen wie vorhanden. Liegt allerdings die Startposition nicht innerhalb des Strings, liefert `substr()` nur `false` zurück.

7.4.2 Whitespaces entfernen

Bei Whitespaces denkt man automatisch an Leerzeichen. Diese fallen durchaus darunter, allerdings gehören auch Zeilenumbruch, Tabulator etc. dazu (im Einzelnen `\n`, `\r`, `\t`, `\v`, `\0`). Wann aber müssen solche Zeichen entfernt werden? Beispielsweise bei der Vollständigkeitsüberprüfung von Formularen, wenn Sie ausschließen möchten, dass der Benutzer nur Leerzeichen eingibt, oder aber wenn Sie Daten sauber, d. h. ohne Whitespaces am Anfang oder am Ende, in die Datenbank speichern möchten.

PHP bietet zum Bereinigen von Strings einige Funktionen:

- `trim(String)` entfernt Whitespaces am Anfang und am Ende des Strings.
- `ltrim(String)` entfernt sie nur auf der linken Seite, also am Anfang des Strings.
- `rtrim(String)` löscht Whitespaces rechts, also am Ende.
- `chop(String)` ist ein Alias von `rtrim()`, entfernt also auch die Whitespaces am Ende eines Strings.

Hier ein kleines Beispiel:

```
<?php
  $a = "   Leerzeichen   ";
  echo "Viele " . trim($a) . "!";
?>
```

Listing 7.17 »trim()« (»trim.php«)

7.5 Suchen und Ersetzen

Suchen und Ersetzen gehören zu den Kategorien, die einem Texteditor zur Ehre gereichen. Ganz so ausgefeilt muss das Suchen und Ersetzen in Strings meist nicht sein. Dennoch bietet PHP eine Vielzahl an Möglichkeiten.

Hinweis

Noch mehr Möglichkeiten erhalten Sie mit regulären Ausdrücken. Mehr dazu in Kapitel 10, »Reguläre Ausdrücke«.

7.5.1 Suchen

Die Suchfunktionen unterscheiden sich in dem, was sie zurückliefern. Ist es die Position des gefundenen String-Teils oder vielleicht der Reststring ab dieser Position? Danach werden in diesem Abschnitt die Suchfunktionen unterteilt.[9]

Position

Für das Suchen einer Position ist in erster Linie die Funktion `strpos(String, Suchstring, Beginn)` zuständig. Sie durchsucht einen `String` nach dem `Suchstring`[10] und liefert die erste (!) Position, an der er auftaucht. Diese Position ist der erste Buchstabe des Suchstrings. Wenn Sie den optimalen Parameter `Beginn` als Integer angeben, beginnt `strpos()` erst an dieser Position mit der Suche.

```
$a = "Die blauen Reiter.";
echo strpos($a, "blau");
```

Diese Codezeilen liefern als Ergebnis `4`, da dies die Startposition des `b` von `blau` ist. Hätten Sie nach `grau` gesucht, hätte PHP `false` zurückgegeben, da kein Suchergebnis vorliegt.

Hinweis

Achten Sie bei Vergleichen darauf, dass Sie mit exakter Gleichheit (`===`) prüfen. Ansonsten erhalten Sie bei einem Positionsergebnis `0`, also einer gefundenen Zeichenkette an erster Position das Ergebnis falsch, obwohl ja etwas gefunden wurde.

Mit `stripos(String, Suchstring, Beginn)` erreichen Sie dasselbe wie mit `strpos()`, nur dass die Suche unabhängig von Groß- und Kleinschreibung ist.

```
$a = "Die blauen Reiter.";
echo stripos($a, "Blau");
```

liefert also auch `4`, obwohl der blaue Reiter mit kleinem `b` beginnt. `stripos()` gibt es allerdings erst in PHP 5. Sie können sie allerdings auch sehr einfach selbst simulieren,

9 Im Prinzip könnte `substr()` bzw. `subistr()` auch unter Suchen eingeordnet werden, nur dass die Rückgabe hier eben der gefundene String selbst ist; eine akademische Diskussion ...

10 Der String und der Suchstring werden in der Onlinedokumentation auch sehr nett als Heuhaufen und Nadel bezeichnet.

indem Sie `String` und `Suchstring` in Klein- oder Großbuchstaben umwandeln, bevor Sie `strpos()` einsetzen:

```
function stripos_php4($string, $suche) {
  return strpos(strtolower($string), strtolower($suche));
}
```

Listing 7.18 »stripos()« für PHP 4 simuliert (»stripos_php4.php«)

Hinweis

`strrpos(String, Suchstring)` ist das Gegenstück zu `strpos()`. Hier verläuft die Suche von hinten nach vorn (zu erkennen am `r` für right im Namen). Das Ergebnis ist also das letzte Vorkommen eines Suchstrings. `strripos(String, Suchstring)` arbeitet wie `strrpos()`, nur dass es nicht zwischen Groß- und Kleinschreibung unterscheidet. `strripos()` gibt es ab PHP 5.0.

Reststring

`strstr(String, Suchstring)` liefert den Rest des Strings ab dem ersten Auftauchen des Suchstrings. Der Suchstring ist im Reststring enthalten.

```
<?php
  $a = "Die blauen Reiter.";
  echo strstr($a, "blau");
?>
```

Listing 7.19 »strstr()« (»strstr.php«)

Hinweis

`strchr(String, Suchstring)` ist das Alias zu `strstr()`. `strrchr(String, Suchstring)` funktioniert wie die beiden, nur dass die Suche am Ende des Strings beginnt. Man könnte es vielleicht vermuten, aber `strrstr()` gibt es nicht.

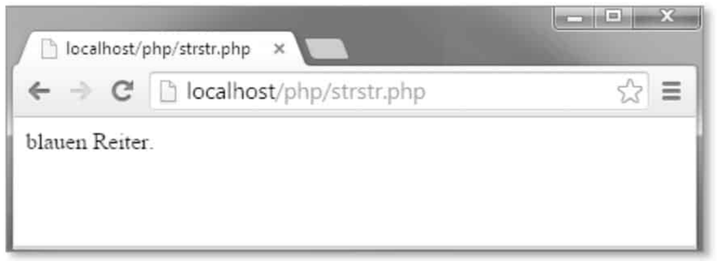


Abbildung 7.12 Traurige Reste: Das »Die« ist verschwunden.

Hinweis

An dem i im Namen ist wieder die von Groß- und Kleinschreibung unabhängige Variante `stristr(String, Suchstring)` zu erkennen. Im Gegensatz zu einigen anderen »i«-Funktionen gibt es sie auch schon in PHP 4 (und sogar PHP 3).

Häufigkeit des Vorkommens

Die Funktion `substr_count(String, Suchstring, Position, Länge)` zählt, wie oft ein `Suchstring` in einem `String` vorkommt. Ab PHP 5.1 gibt es die Parameter `Position`, an der zu suchen begonnen wird, und die `Länge` in Zeichen, auf der gesucht wird.

```
<?php
  $a = "Jippieeehjey";
  echo substr_count($a, "e");
?>
```

Listing 7.20 »substr_count()« (»substr_count.php«)

Das obige Skript meldet vier Vorkommen von `e` in einem längeren Jubel-String.

Alle Positionen finden

Die vorgefertigten Funktionen von PHP helfen meistens, aber nicht immer. Wenn Sie beispielsweise alle Positionen, an denen ein bestimmter Suchstring vorkommt, in einem Array speichern möchten, müssen Sie zur Handarbeit greifen. Das folgende Skript erledigt dies:

```
<?php
  $a = "Jippieeehjey";
  $positionen = array();
  $i = 0;
  $position = strpos($a, "e");
  while ($position != false) {
    $positionen[$i] = $position;
    $position = strpos($a, "e", $position + 1);
    $i++;
  }
  print_r($positionen);
?>
```

Listing 7.21 Alle Positionen in einem Array speichern (»suchen_alle.php«)

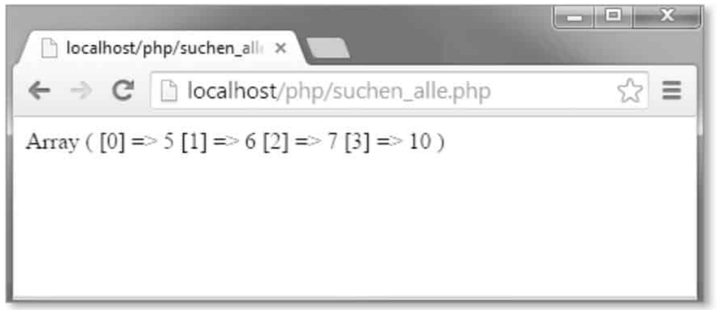


Abbildung 7.13 Das Array mit allen Positionen

Tipp

Wenn Sie eine solche Funktionalität öfter benötigen, schreiben Sie einfach eine eigene Funktion und erstellen eine PHP-Datei damit. Diese Datei binden Sie dann in neue Dateien ein.

Mehrere Zeichen suchen – PHP 5

`strpbrk(String, Zeichen)` ist neu in PHP 5 und erlaubt die Suche nach mehreren Zeichen. Die Zeichen werden hintereinander als String angegeben. Sobald eines der Zeichen gefunden wurde, wird der gesamte String bis ans Ende zurückgeliefert.

Das folgende Skript sucht nach `x`, `b` und dem Punkt:

```
<?php
  $a = "Die blauen Reiter.";
  echo strpbrk($a, "xb.");
?>
```

Listing 7.22 »strpbrk()« (»strpbrk.php«)

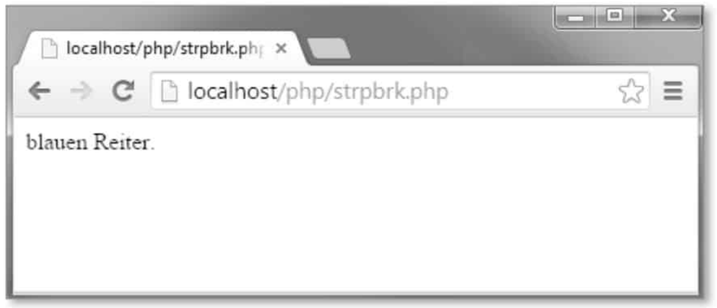


Abbildung 7.14 Als Erstes wird das »b« gefunden und alles danach angezeigt.

7.5.2 Ersetzen

Auch zum Ersetzen von Teilen eines Strings gibt es mehrere Funktionen. Sie unterscheiden sich hauptsächlich darin, wie viel sie ersetzen.

An Position ersetzen

Die Funktion `substr_replace(String, Ersatz, Startposition, Länge)` funktioniert wie der kleine Bruder `substr()`, nur dass der angegebene Bereich nicht ausgeschnitten, sondern ersetzt wird. Die `Länge` ist optional, negative Werte sind sowohl für Startposition als auch für Länge möglich.

Das folgende Skript ersetzt die roten durch blaue Reiter:

```
<?php
  $a = "Die roten Reiter.";
  echo substr_replace($a, "blauen", 4, 5);
?>
```

Listing 7.23 »substr_replace()« (»substr_replace.php«)

Suchen und Ersetzen

Die Funktion `str_replace(Suchstring, Ersatz, String)` stellt die »kleine« Alternative zum Suchen und Ersetzen mit regulären Ausdrücken dar. Sie hat einige Vorteile: Sie ist performant, da sie auf einem Binärvergleich basiert, und sie ist einfach zu merken. Der erste Parameter enthält den `Suchstring`. Der zweite den `Ersatz` für die gefundene Stelle, der dritte ist der `String`, in dem gesucht und ersetzt wird.

> **Hinweis**
>
> Achtung, die Parameterreihenfolge ist bei den String-Funktionen in PHP leider nicht einheitlich. Hier steht der String, um den es geht, beispielsweise am Ende.

```
<?php
  $a = "Jippieeejey";
  echo str_replace("e", "i", $a);
?>
```

Listing 7.24 »str_replace()« (»str_replace.php«)

Im obigen Code werden alle `e` durch `i` ausgetauscht.

> **Tipp**
>
> Das schnelle Suchen und Ersetzen eignet sich beispielsweise auch sehr gut für Platzhalter in Ihrem Code.

Neben einfachen Strings unterstützt `str_replace()` auch Arrays für alle drei Parameter.

```
<?php
  $a = "Jippieeejey";
  $b = "Holadrioe";
  $ergebnis = str_replace(array("e", "o"), array("i", "ö"),
                                                  array($a, $b));
  print_r($ergebnis);
?>
```

Listing 7.25 »str_replace()« mit Arrays (»str_replace_array.php«)

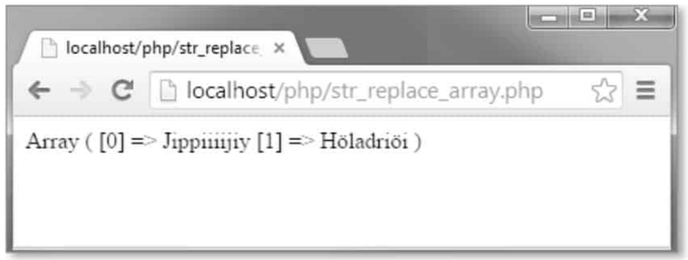


Abbildung 7.15 Wildes Zeichenkettenwechseln ...

> **Hinweis**
>
> `str_ireplace(Suchstring, Ersatz, String)` ist die case-insensitive Variante von `str_replace()` und ansonsten baugleich. Allerdings gibt es die Funktion erst in PHP 5.

Mehrere Zeichen ersetzen

Die Funktion `strtr(String, Von, In)` funktioniert wie `strpbrk()`. Sie sucht nach in einem String angegebenen Zeichen (`Von`) und ersetzt sie durch in einem zweiten String eingetragene Zeichen (`In`).

Hier ein einfaches Beispiel:

```
<?php
  $a = "Jippieeejey";
  echo strtr($a, "ei", "ie");
?>
```

Listing 7.26 »strstr()« (»strstr.php«)

Aus `Jippieeejey` wird `Jeppeiiijiy`.

Hinweis

Wenn nicht gleich viele Zeichen im `Von`-String und im `In`-String vorhanden sind, werden die auf der einen Seite überschüssigen ignoriert.

`strtr(String, Array)` kennt noch eine zweite Syntax mit assoziativem Array. In diesem Fall ist der Index des jeweiligen Array-Elements das `Von` und der Wert das `In`. Das folgende Skript bewirkt also dasselbe wie Listing 7.26, nur eben mit assoziativem Array.

```
<?php
  $a = "Jippieeejey";
  echo strtr($a, array("e"=>"i", "i"=>"e"));
?>
```

Listing 7.27 »strtr()« mit assoziativem Array (»strtr_asso.php«)

7.6 Sonderzeichen, HTML etc.

Egal, ob Sie gerade mit HTML, Datenbankabfragen oder Dateien arbeiten, Sonderzeichen werden Sie immer wieder antreffen. PHP bietet für die wichtigsten Anwendungsgebiete schon fertige Funktionen.

7.6.1 Entwerten – für Datenbanken

Beispielsweise für Datenbankabfragen müssen Sie bestimmte Zeichen entwerten.[11] Dies erfolgt mit dem Backslash (\). Die Funktion `addslashes(String)` fügt vor einfache und doppelte Anführungszeichen, vor Backslash und `null`-Werte einen Backslash ein. Wenn Sie also einen String als Wert in eine Datenbank schreiben, verwenden Sie oftmals diese Funktion, damit die genannten Zeichen von SQL nicht als zur Syntax gehörig interpretiert werden. Allerdings besitzen die meisten Datenbankschnittstellen in PHP eigene Entwertungsfunktionen. Diese sind `addslashes()` im Regelfall vorzuziehen.

Um `addslashes(String)` wieder rückgängig zu machen, verwenden Sie `stripslashes(String)`. Hier ein einfaches Beispiel, das beide einsetzt:

```
<?php
  $a = 'Caesar sagte: "Ich kam, sah und siegte!"';
  $a = addslashes($a);
  echo "Mit Backslash: " . $a;
  $a = stripslashes($a);
  echo "<br />Ohne: " . $a;
?>
```

Listing 7.28 Entwerten mit Backslash (»addslashes.php«)



Abbildung 7.16 Oben mit, unten ohne Backslashs

Etwas flexibler als `addslashes()` ist die Funktion `addcslashes(String, Zeichen)`. Sie fügt zu allen Zeichen Backslashs hinzu, die im String `Zeichen` als Parameter angegeben sind.

```
addcslashes($a, '"n')
```

setzt also vor doppelte Anführungszeichen und vor `n` einen Backslash.

Hinweis

Die Rückumwandlung erfolgt mit `stripcslashes(String)`. Sonderzeichen wie `\n` werden allerdings ignoriert. Das heißt, würden Sie `addcslashes($a, 'n')` einsetzen, hätten Sie nach der Rückumwandlung immer noch statt aller `n` Zeilenumbrüche im Quellcode. Wollen Sie diesen Effekt vermeiden, verwenden Sie `stripslashes()`. Das entfernt alle Backslashs.

7.6.2 Entwerten – für reguläre Ausdrücke

Reguläre Ausdrücke verwenden – ähnlich wie SQL – eigene Sonderzeichen, die deswegen in einem String entwertet werden sollten. Dafür ist die Funktion `quotemeta(String)` zuständig.[12] Folgende Zeichen werden mit einem Backslash entwertet:

```
. \\ + * ? [ ^ ] ( $ )
```

11 Für diesen Vorgang lassen sich übrigens verschiedene Begriffe verwenden. Neben »entwerten« ist auch noch »maskieren« oder sogar »auskommentieren« anzutreffen.

12 Der Begriff *quoten* steht für »entwerten«.

Hier ein simples Beispiel:

```
<?php
  $a = "Ergibt 50 * (5 - 3) 100?";
  echo quotemeta($a);
?>
```

Listing 7.29 »quotemeta()« (»quotemeta.php«)



Abbildung 7.17 Alle relevanten Zeichen sind auskommentiert.

Tipp

Zum Rückumwandeln verwenden Sie einfach `stripslashes()`.

7.6.3 HTML

HTML ist eine besondere oder, man könnte auch sagen, sonderbare Sprache. Was nicht in einen einfachen Zeichensatz passt, kennt der Browser per Definition erst einmal nicht. *Entitäten* oder auf Englisch *Entities*[13] stehen für Sonderzeichen bereit. Darunter fallen beispielsweise auch deutsche Umlaute. PHP bietet einige Funktionen, für den Umgang mit Sonderzeichen im Speziellen.

Hinweis

Aktuell hat sich in den meisten Webapplikationen der Einsatz von UTF-8 als Zeichensatz durchgesetzt. Dank dem Universalzeichensatz kann man generell die Umlautmaskierung weglassen. Dessen ungeachtet ist es immer sinnvoll, die Umwandlung für andere Zeichen wie Anführungszeichen und spitze Klammern vorzunehmen, die in HTML spezielle Bedeutung haben.

13 Eine Entität ist hier als vorgefertigte Zeichenfolge zu verstehen, die als Platzhalter für ein Sonderzeichen steht. Entitäten beginnen in HTML immer mit dem Et-Zeichen und enden mit Strichpunkt. `ä` steht beispielsweise für ä. Das Kürzel ist sogar leicht verständlich: `auml` bedeutet »a Umlaut«.

Sonderzeichen umwandeln

Alle HTML-relevanten Sonderzeichen können Sie in einem String mit der Funktion `htmlentities(String)` umwandeln. Im folgenden Listing stellen wir Umlaute und doppelte Anführungszeichen mit und ohne Umwandlung gegenüber.

```
<?php
  $a = 'Umlaute: "Ä", "ä", "Ö", "ö", "Ü", "ü"';
  echo "Ohne Umwandlung: " . $a . "<br />\n";
  echo "Mit Umwandlung: " . htmlentities($a);
?>
```

Listing 7.30 Mit und ohne Umwandlung (»html.php«)

In einem deutschen Browser werden beide Varianten gleich wiedergegeben. In einem englischen ist das nicht der Fall. Sie sehen den Unterschied allerdings schon im Quellcode (siehe Abbildung 7.18). Aus den Anführungszeichen und den Umlauten wurden HTML-Entitäten.

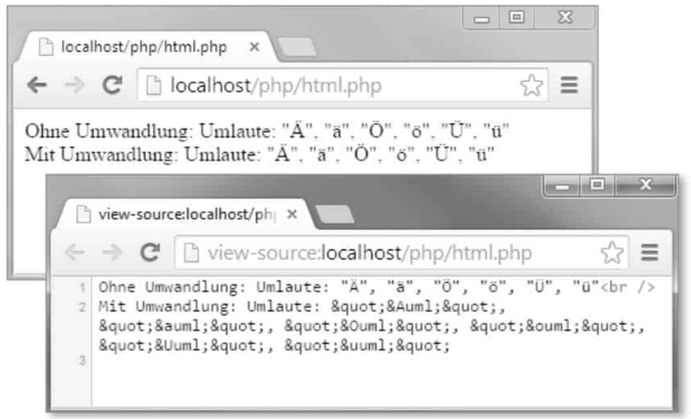


Abbildung 7.18 Umlaute in HTML-Sonderzeichen umwandeln

Tipp

Wenn Sie beispielsweise ein Gästebuch entwickeln, sollten Sie die Benutzereingaben vor der Ausgabe unbedingt in HTML-Sonderzeichen umwandeln. Denn dadurch verschwinden auch eventuell vom Benutzer eingesetzte HTML-Tags, die das Layout der Seite zerstören könnten[14] oder sogar für böswillige Skripte verwendet werden können.[15] Wenn Sie Formatierungen im Gästebuch zulassen möchten, müssen Sie genauer filtern.

14 Im Sinne der Websicherheit spricht man hier von *Defacement*.
15 Dies heißt *Cross-Site-Scripting*.

`htmlentities(String, Anführungszeichen, Zeichensatz)` bietet noch zwei optionale Parameter. Bei `Anführungszeichen` können Sie in einer Konstante setzen, wie doppelte und einfache Anführungszeichen behandelt werden:

- `ENT_QUOTES` wandelt alle Anführungszeichen um.
- `ENT_NOQUOTES` lässt alle Anführungszeichen, wie sie sind.
- `ENT_COMPAT` konvertiert nur doppelte Anführungszeichen in HTML-Sonderzeichen. Dies ist der Standardwert.

```
htmlentities($a, ENT_NOQUOTES)
```

verhindert also, dass Anführungszeichen umgewandelt werden. Statt `$quot;` sehen Sie im Quelltext das doppelte Anführungszeichen.

Der dritte Parameter erlaubt die Konvertierung mit einem festgelegten Zeichensatz. Der Standardwert hierfür ist `ISO-8859-1`. Die weiteren Werte entnehmen Sie der folgenden Tabelle:

Zeichensatz	Alternative Bezeichnungen	Beschreibung
ISO-8859-1	ISO8859-1	Westeuropa, Latin-1
ISO-8859-15	ISO8859-15	Westeuropa, Latin-9. Neu gegenüber ISO-8859-1: Eurozeichen, französische Akzente, finnische Buchstaben
UTF-8		8-Bit-Unicode
cp866	ibm866, 866	kyrillischer Zeichensatz (ab PHP 4.3.2)
cp1251	Windows-1251, win-1251, 1251	kyrillischer Zeichensatz für Windows (ab PHP 4.3.2)
cp1252	Windows-1252, 1252	Windows-Zeichensatz Westeuropa
KOI8-R	koi8-ru, koi8r	russischer Zeichensatz (ab PHP 4.3.2)
BIG5	950	traditionelles Chinesisch (Taiwan)
BIG5-HKSCS		traditionelles Chinesisch mit Hong-Kong-Erweiterung
GB2312	936	einfaches Chinesisch
Shift_JIS	SJIS, 932	Japanisch
EUC-JP	EUCJP	Japanisch

Tabelle 7.1 Zeichensätze

Hinweis

Den Parameter `Anführungszeichen` gibt es seit PHP 4.0.3, den dritten Parameter `Zeichensatz` erst seit PHP 4.1.

`htmlspecialchars(String, Anführungszeichen, Zeichensatz)` funktioniert im Prinzip genau wie `htmlentities()`, konvertiert aber nicht alle HTML-Sonderzeichen. Konvertiert werden:

```
< > ' " &
```

Außen vor bleiben beispielsweise die deutschen Umlaute. Warum gibt es die abgespeckte Version? Gerade im englischsprachigen Raum ist die vollständige Umwandlung aller Sonderzeichen oft nicht erforderlich oder nicht erwünscht. Die von `htmlspecialchars()` umgewandelten Entitäten sind gleichzeitig die entscheidenden Zeichen der HTML-Syntax.

Sonderzeichen zurückverwandeln

Nun zum umgekehrten Weg: HTML-Sonderzeichen in einen normalen String zurückverwandeln. Dafür bietet PHP die Funktion `html_entity_decode(String, Anführungszeichen, Zeichensatz)`. Sie verwendet dieselben Parameter wie `htmlentities()`.

```
<?php
  $a = "&lt;p&gt;Text im Absatz mit&lt;br /&gt; Zeilenumbruch&lt;/p&gt;";
  echo html_entity_decode($a);
?>
```

Listing 7.31 »html_entity_decode()« (»html_entity_decode.php«)

Umwandlungstabelle

Der Umwandlung mit `htmlentities()` bzw. `htmlspecialchars()` liegt eine in PHP gespeicherte Tabelle zugrunde. Sie können diese Tabelle mit der Funktion `get_html_translation_table(Version, Anführungszeichen)` auslesen, um zu sehen, was passiert. Die zwei Parameter sind optional: Wenn Sie `Version` weglassen, werden die Einträge für `htmlspecialchars()` geliefert (entspräche dem Wert `0`), wenn Sie `1` eintragen, wird die – deutlich längere – Liste für `htmlentities()` zurückgegeben. Das Rückgabeformat ist ein assoziatives Array. Schlüssel ist das Original, Wert das Ziel der Umwandlung. Beim dritten Parameter `Anführungszeichen` wählen Sie aus den drei schon bekannten Optionen (siehe Abschnitt »Sonderzeichen umwandeln«).

```
<?php
  $tabelle = get_html_translation_table();
  foreach ($tabelle as $schluessel => $wert) {
```

```
    echo htmlentities($schluessel) . " wird zu " . htmlentities($wert) . "<br />";
  }
?>
```

Listing 7.32 »get_html_translation_table()« (»html_tabelle.php«)

Hinweis

Die zurückgegebenen Schlüssel und Werte maskieren wir hier mit `htmlentities()`, damit sie im Browser so dargestellt werden, wie sie sind.

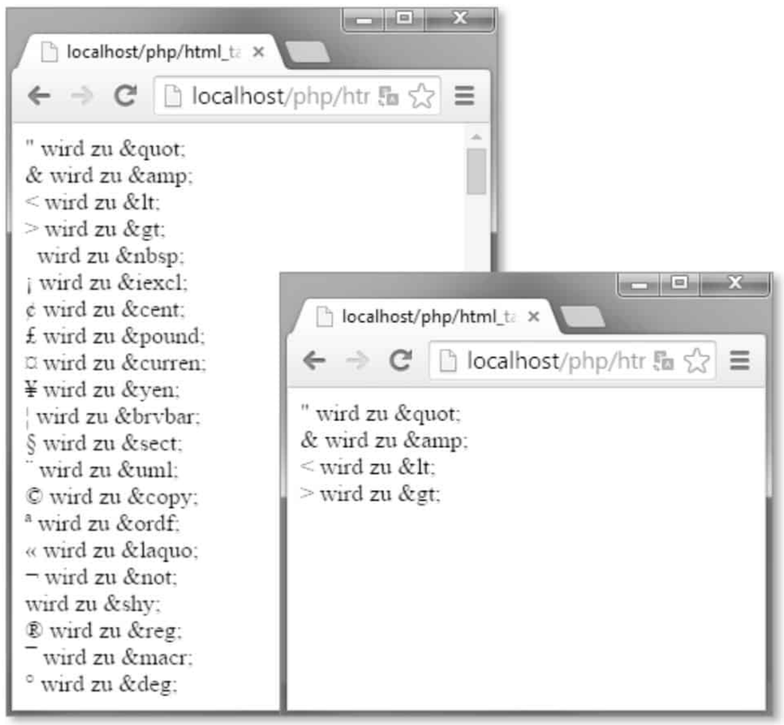


Abbildung 7.19 Die Liste für »htmlentities()« (links) und für »htmlspecialchars()« (rechts)

Tags entfernen

Die Funktion `strip_tags(String, Geschützt)` streicht PHP- und HTML-Tags aus einem String ersatzlos. Im String `Geschützt` tragen Sie die Tags hintereinander ein, die Sie vor dem Ersetzen retten möchten. Vorsicht, die Tags dürfen nicht in XHTML-Schreibweise sein, immer nur das öffnende Tag ist möglich! Groß- und Kleinschreibung macht dagegen keinen Unterschied.

Im folgenden Beispiel wird der Absatz (`<p>` und `</p>`) entfernt, der Zeilenumbruch (`<br />`) bleibt dagegen erhalten:

```
$a = "<p>Text im Absatz mit<br /> Zeilenumbruch</p>";
echo strip_tags($a, "<BR>");
```

7.6.4 URLs

Eine URL, d. h. eine Webadresse (auch *Uniform Resource Locator*), erlaubt es, zusätzliche Informationen anzuhängen. Diese Informationen folgen nach dem Dateinamen und einem Fragezeichen. Dafür gibt es allerdings ein bestimmtes Format.[16] Um dieses Format zu erzeugen, verwenden Sie `urlencode(String)`, um eine URL zurückzuverwandeln, verwenden Sie `urldecode(URL)`.

```
<?php
  $galileo = "http://www.galileo-press.de/index.php?";
  $anhang = "Index 1=Wert 1&Index 2=Wert 2";
  echo $galileo . urlencode($anhang);
?>
```

Listing 7.33 »urlencode()« (»urls.php«)



Abbildung 7.20 Der Anhang hat ein URL-kompatibles Format.

Tipp

In der Praxis benötigen Sie dies, wenn Sie längere Strings, z. B. Benutzereingaben, an die URL anhängen möchten.

16 Alphanumerische Zeichen werden zu %, gefolgt von einem zwei Zeichen langen hexadezimalen Code. Leerzeichen werden zu Pluszeichen.

URLs auseinandernehmen

Wenn Sie die Einzelteile einer URLs benötigen, z. B. Domain, Hostname etc., können Sie `parse_url(URL)` verwenden. Sie übergeben die URL einfach als String und erhalten ein assoziatives Array mit allen enthaltenen Bestandteilen.

```
<?php
  $url = "http://www.galileo-press.de/index.php?
Index+1%3DWert+1%26Index+2%3DWert+2";
  $einzelteile = parse_url($url);
  print_r($einzelteile);
?>
```

Listing 7.34 »parse_url()« (»parse_url.php«)



Abbildung 7.21 Das Array mit der URL in Einzelteilen

7.7 Vergleichen

Den einfachen String-Vergleich kennen Sie bereits aus Kapitel 5, »Programmieren«. Mit exakter Gleichheit und exakter Ungleichheit können Sie sogar den Datentyp mit prüfen.

```
4 === "4"
```

ergibt also `false`.

Die String-Funktionen von PHP halten nun noch einige Vergleiche bereit, die über diese grundlegende Prüfung hinausgehen.

7.7.1 Vergleichsfunktionen

Die Funktionen `strcmp(String1, String2)` und `strcasecmp(String1, String2)` dienen dazu, einen Binärvergleich mit zwei Strings durchzuführen. Der einzige Unterschied

besteht darin, dass `strcasecmp()` nicht zwischen Groß- und Kleinschreibung unterscheidet. Im Gegensatz zum Vergleich mit den entsprechenden Vergleichsoperatoren liefern die Funktionen als Ergebnis, welcher String größer ist. Ist `String1` kleiner, erhalten Sie als Rückgabe einen Wert kleiner 0, sind beide Strings gleich, erhalten Sie `0`, ist `String1` größer, erhalten Sie einen Wert größer 0.

Hinweis

Seit PHP 5.1 gibt es eine Vergleichsfunktion, die nur Teile von zwei Strings miteinander vergleicht: `substr_compare(String1, String2, Start, Länge, Case-sensitiv)`. Die Pflichtparameter sind die zwei Strings und die Position, ab der verglichen werden soll. Die `Länge`, also die Zahl der Zeichen, die verglichen werden soll, ist optional, seit PHP 5.6 ist dafür auch der Wert `0` erlaubt. Ebenso optional ist der boolesche Parameter `Case-sensitiv`, der die Unterscheidung zwischen Groß- und Kleinschreibung mit `false` ausschaltet (Standardwert ist `true`).

7.7.2 Ähnlichkeiten und Unterschiede

Nicht nur wenn man feststellen möchte, ob ein fauler Autor (Schüler, Student, Professor etc.) Text geklaut hat, ist es sinnvoll, Ähnlichkeit bzw. Unterschiede zwischen Strings zu quantifizieren.

Ähnlichkeit

Die Funktion `similar_text(String1, String2, Prozent)` berechnet die Ähnlichkeit zwischen zwei Strings. Optional können Sie eine Variable (als Referenz) für den dritten Parameter angeben. In diese Variable schreibt die Funktion dann das Ergebnis des Vergleichs als Prozentwert. Der Rückgabewert der Funktion ist etwas weniger aussagekräftig als die Prozentangabe, er bezeichnet, wie viele Buchstaben als gleich erkannt werden.

```
<?php
  $a = "PHP ist machtvoll";
  $b = "Alle Macht PHP!";
  $e;
  echo "Wert: " . similar_text($a, $b, $e) . "<br />";
  echo "Prozent: " . $e;
?>
```

Listing 7.35 »similar_text()« (»similar_text.php«)

Die beiden Strings aus dem Listing sind zu 31,25 % ähnlich.

Unterschiede

Ähnlich wie `similar_text()` arbeitet `levenshtein(String1, String2)`. Auch hier werden zwei Strings – dieses Mal nach einem Algorithmus von Levenshtein – verglichen und die Distanz, d. h. der Unterschied zwischen den Strings, geliefert. Die Strings dürfen allerdings nur maximal 255 Zeichen lang sein.

7.7.3 Aussprache

Wenn ein Mensch einen Wert in ein Textfeld schreibt, kann seine Rechtschreibung falsch sein. Für den Programmierer ist das unangenehm, denn er muss viele Vertippmöglichkeiten berücksichtigen. Viele Rechtschreibfehler entstehen allerdings dadurch, dass Menschen schreiben, wie sie sprechen. Hier bietet PHP mit der `soundex(String)` eine gute Möglichkeit, zwei Strings darauf zu vergleichen, ob sie sich ähnlich bzw. gleich anhören.[17]

Wäre also beispielsweise die Eingabe von `Guten Morgen` gefordert, würde ein normaler String-Vergleich liefern, dass `Gutn Morgän` nicht gleich ist. `soundex()` dagegen liefert für beide den gleichen Schlüssel:

```
<?php
  $a = "Guten Morgen";
  $b = "Gutn Morgän";
  echo $a . ": " . soundex($a) . "<br />";
  echo $b . ": " . soundex($b);
?>
```

Listing 7.36 »soundex()« (»soundex.php«)

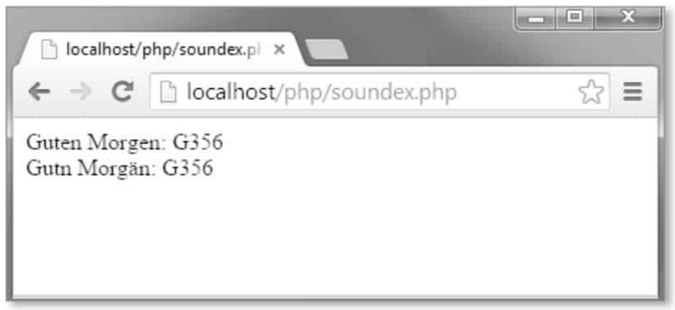


Abbildung 7.22 Die zwei Ausdrücke klingen fast gleich.

17 Die Funktion basiert laut PHP-Dokumentation auf einem Soundex-Algorithmus von Donald Knuth aus »The Art of Computer Programming«, Teil 3, »Sortieren und Suchen«, S. 391 f., erschienen 1973. Ursprünglich geht Soundex allerdings auf einen Algorithmus zurück, der bereits 1918 patentiert wurde, und zwar von Robert C. Russell als US-Patent am 2. April mit der Registernummer 1,261,167.

7.8 Hilfreiches und Nützliches

Dieser Abschnitt ist das Sammelbecken für alle Funktionen, die zu wichtig sind, um ohne Beispiel zu bleiben, auf der anderen Seite aber zu keinem der Hauptthemen gut passen.

7.8.1 ASCII und Umwandlung

Computer speichern Daten als Bytes. Ein Byte hat Werte zwischen 0 und 255. Der ASCII-Zeichencode stellt Buchstaben und Zeichen innerhalb dieses Wertebereichs dar. Das heißt, wenn ein Byte einen Zahlenwert hat, kann per Zahlencode festgestellt werden, welches Zeichen dazu passt. ASCII ist zwar ein alter Zeichensatz, und mittlerweile werden auch in Browsern und HTML-Dokumenten eher die ISO-Zeichensätze eingesetzt, den ASCII-Code von Zeichen oder Buchstaben benötigen Sie allerdings dennoch manchmal. PHP bietet dafür zwei Funktionen:

- `chr(ASCII)` verwandelt einen ASCII-Code in das zugehörige Zeichen.
- `ord(Zeichen)` liefert den ASCII-Code zum zugehörigen Zeichen.

```
<?php
  echo "Zeichen: " . chr(65) . "<br />";
  echo "ASCII: " . ord(A);
?>
```

Listing 7.37 »chr()« und »ord()« (»chr_ord.php«)

7.8.2 Verschlüsselung

Zur Verschlüsselung bietet PHP mehrere Funktionen für unterschiedliche Verschlüsselungstechnologien:

- `md5(String)` berechnet den MD5-Hash[18], eine hexadezimale Zahl mit 32 Zeichen Länge, für einen String.
- `md5_file(Dateiname)` berechnet den MD5-Hash aus einem Dateinamen.

> **Hinweis**
>
> In PHP 5 gibt es für beide Funktionen als zweiten Parameter einen Boolean, der festlegt, ob die Rückgabe als Zahl (`false`, Standardwert) oder als binäre Daten mit 16 Zeichen Länge (`true`) erfolgen soll.

18 Standardisiert ist der zugehörige Algorithmus unter *http://www.ietf.org/rfc/rfc1321.txt*. Ursprünglich erfunden wurde er von Ronald L. Rivest, einem Professor am MIT.

- `crypt(String)` erzeugt die DES-Verschlüsselung eines Strings. Je nach System kommen unterschiedliche Algorithmen zum Einsatz. Allerdings handelt es sich immer um eine Einwegverschlüsselung, die Entschlüsselung ist nicht möglich.
- `sha1(String)` und `sha1_file(Dateiname)` arbeiten wie `md5()` und `md5_file()`, nur verschlüsseln sie stattdessen mit dem US-Secure-Hash-Algorithmus Nummer 1[19].
- `hash(Algorithmus, String)` gibt es seit PHP 5.1.2. Damit kann man beispielsweise *sha256* berechnen. Geben Sie dazu einfach den Algorithmus als String an. Hier steht übrigens auch *md5* zur Verfügung.

> **Tipp**
>
> Mehr, d. h. sicherere und anpassbarere Verschlüsselungen, erhalten Sie mit der *mcrypt*-Bibliothek (*http://www.php.net/manual/de/ref.mcrypt.php*).

Anwendung: Eindeutige ID

Eine eindeutige ID benötigen Sie beispielsweise, wenn Sie ein eigenes Session-Management zur Identifikation Ihrer Benutzer erstellen oder irgendein anderes Element eindeutig kennzeichnen möchten. Zur Berechnung von eindeutigen IDs gibt es viele Ideen und Skripte. Einer der besten und kürzesten Vorschläge kommt vom PHP-Mitentwickler Sterling Hughes:

```
$uid = md5(uniqid(microtime(), 1));
```

Mit der Funktion `uniqid()` wird mittels des aktuellen Datums in Sekunden eine ID berechnet. Anschließend wird daraus noch der 32-stellige MD5-Hash erstellt. Wenn Sie dann noch die minimale Chance ausschließen möchten, dass ein anderer Server die gleiche ID produziert, binden Sie die eindeutige Rechner-ID noch in den String ein:

```
$uid = substr($uid, 0, 16) . getmypid() . substr($uid, 16, 16);
```

Hier das komplette Skript:

```
<?php
  $uid = md5(uniqid(microtime(), 1));
  $uid = substr($uid, 0, 16) . getmypid() . substr($uid, 16, 16);
  echo $uid;
?>
```

Listing 7.38 Eine eindeutige ID (»eindeutigeid.php«)

19 Auch hier Details unter *http://www.ietf.org/rfc/rfc3174.txt*.

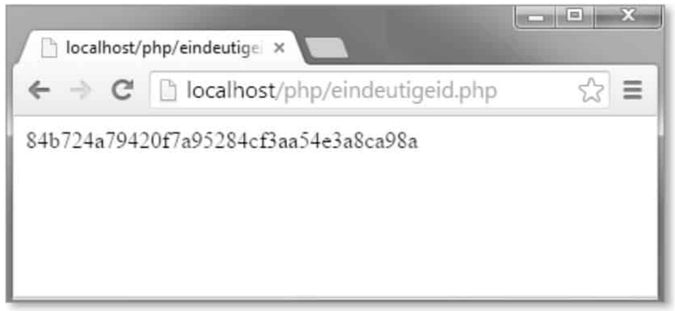


Abbildung 7.23 Die eindeutige ID

7.8.3 Umdrehen

`strrev(String)` dreht einen String um. Selten gebraucht, aber manchmal ganz praktisch:

```
<?php
  $a = "PHP ist toll";
  echo strrev($a)
?>
```

Listing 7.39 »strrev()« (»strrev.php«)

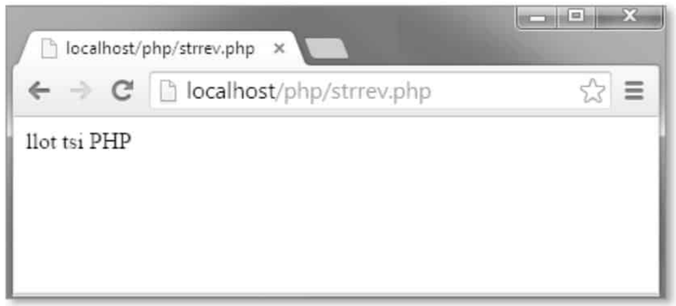


Abbildung 7.24 Ein umgedrehter String

Auf einen Blick

TEIL I Vorbereitungen 29

TEIL II Einstieg in PHP 91

TEIL III Webtechniken 391

TEIL IV Datenbanken 527

TEIL V Kommunikation 717

TEIL VI Fremdformate 801

TEIL VII Administration und Sicherheit 879

TEIL VIII PHP erweitern 963

Inhalt

Vorwort 23

TEIL I Vorbereitungen

1 Allgemeines zu PHP 31

1.1 Geschichte von PHP 31
1.2 Erfolg und Einsatz 32
1.3 Das Konzept von PHP 33
1.4 Neuerungen in PHP 5.1 35
1.5 Neuerungen in PHP 5.3 37
1.6 Die wichtigsten Neuerungen in PHP 5.4, 5.5 und 5.6 38
1.7 Versionen und Anlaufstelle 39

2 Installation 41

2.1 PHP installieren 41
2.1.1 Aufbau von PHP 43
2.1.2 Windows 44
2.1.3 Mac OS X 58
2.1.4 Linux 64
2.2 PEAR installieren 72
2.2.1 PEAR installieren 72
2.2.2 PEAR2 75
2.2.3 PEAR-Pakete ohne Installation 77

3 Test und Hilfe 79

3.1 Häufige Fehler 79
3.1.1 Die Seite kann nicht angezeigt werden 79
3.1.2 Die Webseite wurde nicht gefunden/File not found 81

3.1.3 Server nicht gefunden ... 82
3.1.4 Unable to initialize module ... 83
3.1.5 Modul nicht gefunden ... 84
3.1.6 Der Browser öffnet ein Downloadfenster ... 85
3.1.7 No input file specified (oder so ähnlich) ... 86
3.1.8 Call to undefined function ... 87
3.1.9 Internal Server Error ... 88

3.2 Hilfsquellen ... 89

TEIL II Einstieg in PHP

4 Grundlagen der Sprache 93

4.1 PHP in HTML ... 93
4.1.1 Kommentare ... 95
4.1.2 Anweisungen ... 95
4.1.3 Externe Datei ... 96

4.2 Ausgabe mit PHP ... 100
4.2.1 Anführungszeichen ... 101

4.3 Variablen ... 102
4.3.1 Datentypen ... 103
4.3.2 Benennung ... 106
4.3.3 Variable Variablen ... 106
4.3.4 Variablen ausgeben ... 107
4.3.5 Nützliches und Hilfreiches ... 109
4.3.6 Vordefinierte Variablen ... 114

4.4 Konstanten ... 114

5 Programmieren 117

5.1 Operatoren ... 117
5.1.1 Arithmetische Operatoren ... 117
5.1.2 Vergleichsoperatoren ... 121
5.1.3 Logische Operatoren ... 128
5.1.4 Binärzahlen und bitweise Operatoren ... 129
5.1.5 Operatoren, die aus der Reihe tanzen ... 134
5.1.6 Rangfolge der Operatoren ... 136

5.2 Fallunterscheidungen ... 138
5.2.1 »if« ... 138
5.2.2 »switch« ... 145

5.3 Schleifen ... 149
5.3.1 »for« ... 149
5.3.2 »while« ... 154
5.3.3 »do-while« ... 160

5.4 Sprünge ... 161

6 Funktionen und Sprachkonstrukte 163

6.1 Funktionen ... 163
6.1.1 Parameter ... 164
6.1.2 Gültigkeit von Variablen ... 168
6.1.3 Rückgabewert ... 170
6.1.4 Funktionsnamen in Variablen ... 173
6.1.5 Anonyme Funktionen ... 174
6.1.6 Rekursive Funktionen ... 175
6.1.7 Hilfreiches und Nützliches ... 178
6.1.8 Funktionen von PHP ... 181

6.2 Sprachkonstrukte ... 182
6.2.1 Ausgabe ... 184

7 Strings 191

7.1 Verbinden ... 191

7.2 Teilen und Zusammenfügen ... 192
7.2.1 Auf Zeichen zugreifen ... 192
7.2.2 Gleichmäßig unterbrechen ... 192
7.2.3 Zeilenumbrüche ... 194
7.2.4 Teilen in Strings ... 195
7.2.5 Strings und Arrays ... 197

7.3 Groß- und Kleinschreibung ... 202

7.4 Beschneiden ... 205
7.4.1 Zeichen ausschneiden ... 206
7.4.2 Whitespaces entfernen ... 207

7.5 Suchen und Ersetzen ... 207
7.5.1 Suchen ... 208
7.5.2 Ersetzen ... 212
7.6 Sonderzeichen, HTML etc. ... 214
7.6.1 Entwerten – für Datenbanken ... 214
7.6.2 Entwerten – für reguläre Ausdrücke ... 215
7.6.3 HTML ... 216
7.6.4 URLs ... 221
7.7 Vergleichen ... 222
7.7.1 Vergleichsfunktionen ... 222
7.7.2 Ähnlichkeiten und Unterschiede ... 223
7.7.3 Aussprache ... 224
7.8 Hilfreiches und Nützliches ... 225
7.8.1 ASCII und Umwandlung ... 225
7.8.2 Verschlüsselung ... 225
7.8.3 Umdrehen ... 227

8 Arrays 229

8.1 Grundlagen ... 229
8.1.1 Arrays erstellen ... 229
8.1.2 Elemente hinzufügen und ändern ... 231
8.1.3 Elemente löschen ... 232
8.1.4 Assoziative Arrays ... 233
8.1.5 Kurzschreibweise ab PHP 5.4 ... 233
8.1.6 Multidimensionale Arrays ... 234
8.2 Arrays und Schleifen ... 234
8.2.1 »for« ... 234
8.2.2 »foreach« ... 235
8.2.3 Funktionen zur Iteration ... 238
8.3 Untersuchen ... 240
8.4 Transformieren ... 241
8.4.1 Hinzufügen und Entfernen ... 241
8.4.2 Löschen und Ersetzen ... 242
8.4.3 Verbinden ... 244
8.4.4 Variablen und Arrays ... 246

8.4.5 Dereferenzierung ... 248
8.4.6 Umwandlung in Parameter ... 249
8.5 Suchen und Sortieren ... 250
8.5.1 Suchen ... 250
8.5.2 Sortieren ... 251
8.6 Superglobale Arrays ... 252

9 Mathematische und Datumsfunktionen 259

9.1 Mathe ... 259
9.1.1 Basics ... 259
9.1.2 Konstanten ... 260
9.1.3 Zahlen konvertieren ... 261
9.1.4 Zufallszahlen ... 264
9.1.5 Maximal, minimal und Runden ... 265
9.1.6 Bogenmaß und mehr ... 267
9.1.7 Höhere Genauigkeit ... 267
9.2 Datum ... 270
9.2.1 Aktuelles Datum ... 270
9.2.2 »DateTime« ... 275
9.2.3 Beliebige Datumswerte ... 276
9.2.4 Zeitstempel ... 277
9.2.5 Datum formatieren ... 282
9.2.6 Countdown – mit Daten rechnen ... 286

10 Reguläre Ausdrücke 291

10.1 Grundlagen ... 291
10.2 POSIX ... 293
10.3 Perl-kompatibel ... 295
10.4 Anwendungsbeispiele ... 300
10.4.1 Postleitzahlen ... 300
10.4.2 Telefon- und Faxnummern ... 302
10.4.3 Links filtern ... 302

11 Objektorientiert programmieren 305

11.1 Klassen und Objekte – Grundbegriffe ... 305
11.1.1 Klassen und Objekte in PHP ... 307
11.1.2 Eigenschaften ... 308
11.1.3 Methoden ... 310
11.1.4 Vererbung ... 315

11.2 Fortgeschrittenes ... 319
11.2.1 Wert oder Referenz ... 319
11.2.2 Konstruktor ... 320
11.2.3 Destruktor ... 321
11.2.4 Privat, geschützt etc. ... 321
11.2.5 Interfaces ... 328
11.2.6 Late Static Binding ... 330
11.2.7 Abstrakte Klassen ... 331
11.2.8 Konstanten ... 333
11.2.9 Überladen ... 333
11.2.10 Traits ... 339

11.3 Hilfreiches und Nützliches ... 342
11.3.1 Fehlerlevel ... 343
11.3.2 »__autoload()« ... 344
11.3.3 »__METHOD__« ... 345
11.3.4 »__toString()« ... 347
11.3.5 Klassentypen und »instanceof« ... 347
11.3.6 Objekte vergleichen ... 349
11.3.7 Objekte serialisieren ... 350
11.3.8 Objekte automatisiert auslesen ... 353
11.3.9 Iteration ... 355
11.3.10 Reflection API ... 357
11.3.11 SPL ... 359

11.4 Unterschiede zwischen PHP 4 und 5 ... 368

12 Namespaces 371

12.1 Warum Namespaces? ... 372

12.2 Mit Namespaces arbeiten ... 373
12.2.1 Namespaces verwenden ... 374

12.2.2 Den aktuellen Namespace ermitteln ... 375
12.2.3 Namespaces per Alias ... 376

13 Entwurfsmuster: MVC & Co. 379

13.1 Zend Framework ... 380

13.2 MVC/MPS ... 382

13.3 Singleton/Einzelstück ... 387

13.4 Factory/Fabrik ... 389

TEIL III Webtechniken

14 Formulare 393

14.1 Vorbereitungen ... 393

14.2 Formulare mit PHP ... 396
14.2.1 Die gute alte Zeit ... 397
14.2.2 Versandmethoden ... 400
14.2.3 Textfeld(er) ... 403
14.2.4 Radiobuttons ... 405
14.2.5 Checkboxen ... 406
14.2.6 Auswahllisten ... 408
14.2.7 Versand feststellen ... 411

14.3 Formularvalidierung ... 415
14.3.1 Textfeld(er) ... 415
14.3.2 Radiobuttons ... 417
14.3.3 Checkboxen ... 418
14.3.4 Auswahllisten ... 420
14.3.5 Detailliertere Fehlermeldung ... 424

14.4 Vorausfüllung ... 429
14.4.1 Vorbereitungen ... 430
14.4.2 Textfelder ... 430
14.4.3 Radiobuttons ... 432
14.4.4 Checkboxen ... 433
14.4.5 Auswahllisten ... 433

14.5 Dateiuploads ... 442
14.6 Anwendungsbeispiele ... 445
14.6.1 JavaScript-Formularprüfung ... 445
14.6.2 Bildergalerie ... 450
14.7 Einstellungen ... 453

15 Cookies 455

15.1 Vorbereitungen ... 455
15.2 Fakten und Hintergründe ... 457
15.2.1 Was ist ein Cookie? ... 458
15.2.2 Einschränkungen ... 458
15.2.3 Der gläserne Surfer? ... 459
15.3 Mit Cookies in PHP arbeiten ... 460
15.3.1 Cookies setzen ... 460
15.3.2 Cookies auslesen ... 466
15.3.3 Cookies löschen ... 469
15.3.4 »Neue« Cookies ... 473
15.4 Cookie-Test ... 475
15.5 Abschließende Überlegungen ... 477

16 Sessions 481

16.1 Vorbereitungen ... 482
16.2 Fakten, Hintergründe und Konfiguration ... 483
16.2.1 Daten behalten ... 483
16.2.2 Performance ... 485
16.3 Mit Sessions in PHP arbeiten ... 486
16.3.1 Daten schreiben ... 487
16.3.2 Daten auslesen ... 489
16.3.3 Daten löschen ... 490
16.3.4 Daten behalten ... 492
16.4 Geschützter Bereich ... 493
16.5 Sessions in Datenbanken ... 495
16.6 Sicherheitsbedenken ... 503

17 E-Mail 507

17.1 Vorbereitungen ... 507
17.2 Mails mit PHP versenden ... 512
17.2.1 Standardmails ... 512
17.2.2 MIME-Mails ... 517
17.2.3 IMAP und POP ... 525

TEIL IV Datenbanken

18 SQL 529

18.1 Datenbanken und Tabellen anlegen ... 530
18.1.1 Primärschlüssel ... 531
18.1.2 Datentypen ... 531
18.2 Daten eintragen ... 533
18.3 Daten abfragen ... 534
18.4 Daten aktualisieren ... 537
18.5 Daten löschen ... 538
18.6 Besonderheiten ... 538
18.6.1 Relationales Datenbankdesign ... 538
18.6.2 Joins ... 540
18.6.3 Aggregatfunktionen ... 541
18.6.4 Transaktionen ... 543
18.6.5 Stored Procedures ... 544

19 PDO 547

19.1 Vorbereitungen ... 548
19.2 Datenbankzugriff mit PDO ... 550
19.2.1 Verbindungsaufbau ... 550
19.2.2 Abfragen ... 551
19.2.3 Rückgabewerte ... 554
19.2.4 Besonderheiten ... 556

19.3 Gästebuch ... 557
19.3.1 Tabelle anlegen ... 557
19.3.2 Daten eintragen ... 558
19.3.3 Daten ausgeben ... 560
19.3.4 Daten löschen ... 561
19.3.5 Daten bearbeiten ... 564

20 MySQL 567

20.1 Vorbereitungen ... 568
20.2 Datenbankzugriff mit MySQL ... 574
20.2.1 Verbindungsaufbau ... 575
20.2.2 Abfragen ... 576
20.2.3 Rückgabewerte ... 580
20.2.4 Besonderheiten ... 585
20.3 Alte MySQL-Versionen ... 589
20.3.1 Verbindungsaufbau ... 590
20.3.2 Abfragen ... 590
20.3.3 Rückgabewerte ... 591
20.4 Anwendungsbeispiel ... 592
20.4.1 Tabelle anlegen ... 592
20.4.2 Daten eintragen ... 593
20.4.3 Daten ausgeben ... 595
20.4.4 Daten löschen ... 597
20.4.5 Daten bearbeiten ... 599
20.5 Einstellungen ... 602

21 SQLite 603

21.1 Vorbereitungen ... 604
21.2 Datenbankzugriff mit SQLite ... 605
21.2.1 Verbindungsaufbau ... 605
21.2.2 Abfragen ... 606
21.2.3 Rückgabewerte ... 609
21.2.4 Besonderheiten ... 612

21.3 Anwendungsbeispiel ... 616
21.3.1 Tabelle anlegen ... 616
21.3.2 Daten eintragen ... 617
21.3.3 Daten ausgeben ... 619
21.3.4 Daten löschen ... 620
21.3.5 Daten bearbeiten ... 622

22 Microsoft SQL Server 625

22.1 Vorbereitungen ... 625
22.2 Microsoft SQL Server Driver for PHP ... 627
22.2.1 Verbindungsaufbau ... 629
22.2.2 Abfragen ... 630
22.2.3 Rückgabewerte ... 631
22.2.4 Besonderheiten ... 632
22.3 Anwendungsbeispiel ... 637
22.3.1 Tabelle anlegen ... 637
22.3.2 Daten eintragen ... 638
22.3.3 Daten ausgeben ... 639
22.3.4 Daten löschen ... 641
22.3.5 Daten bearbeiten ... 643

23 Oracle 647

23.1 Vorbereitungen ... 647
23.2 Datenbankzugriff mit Oracle ... 652
23.2.1 Verbindungsaufbau ... 652
23.2.2 Abfragen ... 653
23.2.3 Rückgabewerte ... 656
23.2.4 Besonderheiten ... 659
23.3 Anwendungsbeispiel ... 661
23.3.1 Tabelle anlegen ... 662
23.3.2 Daten eintragen ... 662
23.3.3 Daten ausgeben ... 664
23.3.4 Daten löschen ... 665
23.3.5 Daten bearbeiten ... 667

24 PostgreSQL 671

24.1 Vorbereitungen ... 671

24.2 Datenbankzugriff mit PostgreSQL ... 675
- 24.2.1 Verbindungsaufbau ... 675
- 24.2.2 Abfragen ... 676
- 24.2.3 Rückgabewerte ... 678
- 24.2.4 Besonderheiten ... 680

24.3 Anwendungsbeispiel ... 687
- 24.3.1 Tabelle anlegen ... 687
- 24.3.2 Daten eintragen ... 688
- 24.3.3 Daten ausgeben ... 690
- 24.3.4 Daten löschen ... 691
- 24.3.5 Daten bearbeiten ... 693

24.4 Einstellungen ... 695

25 MongoDB 697

25.1 Vorbereitungen ... 697

25.2 Datenbankzugriff mit MongoDB ... 701
- 25.2.1 Verbindungsaufbau ... 702
- 25.2.2 Einfügen ... 703
- 25.2.3 Abfragen und Rückgabewerte ... 704
- 25.2.4 Aktualisieren ... 706
- 25.2.5 Löschen ... 707

25.3 Anwendungsbeispiel ... 708
- 25.3.1 Datenspeicher anlegen ... 708
- 25.3.2 Daten eintragen ... 708
- 25.3.3 Daten ausgeben ... 710
- 25.3.4 Daten löschen ... 711
- 25.3.5 Daten bearbeiten ... 713

25.4 Einstellungen ... 715

TEIL V Kommunikation

26 Dateien 719

26.1 Vorbereitungen ... 719

26.2 Dateihandling mit PHP ... 720
- 26.2.1 Mit Dateien arbeiten ... 721
- 26.2.2 Mit dem Dateisystem arbeiten ... 726

26.3 Anwendungsbeispiele ... 731
- 26.3.1 Gästebuch ... 732
- 26.3.2 Dateibrowser ... 737

26.4 Einstellungen ... 739

27 Verbindung nach außen 741

27.1 Vorbereitungen ... 741

27.2 Verbindung nach außen mit PHP ... 742
- 27.2.1 Streams ... 742
- 27.2.2 HTTP-Streams ... 744
- 27.2.3 PHP-Streams ... 749
- 27.2.4 Kompressions-Streams ... 753

27.3 Anwendungsbeispiele ... 756
- 27.3.1 Textversion von Webseiten ... 756
- 27.3.2 Onlinekomprimierer ... 758

28 Web Services 761

28.1 Vorbereitungen ... 761
- 28.1.1 Web-Services-Grundlagen ... 761
- 28.1.2 Installation ... 768

28.2 nuSOAP ... 768
- 28.2.1 Server ... 769
- 28.2.2 Client ... 771
- 28.2.3 WSDL ... 772
- 28.2.4 Fazit ... 776

28.3 PHP-SOAP ... 776
28.3.1 Server ... 777
28.3.2 Client ... 778
28.3.3 WSDL ... 779
28.3.4 Fazit ... 781
28.4 REST ... 781
28.5 UDDI ... 784

29 JavaScript 787

29.1 Vorbereitungen ... 788
29.2 JavaScript mit PHP verbinden ... 788
29.2.1 PHP-Variablen mit JavaScript auslesen ... 789
29.2.2 JavaScript-Variablen mit PHP auslesen ... 793
29.3 Ajax ... 793

TEIL VI Fremdformate

30 XML 803

30.1 Vorbereitungen ... 803
30.1.1 XML-Grundlagen ... 803
30.1.2 Installation ... 809
30.2 XML-Zugriff ... 809
30.2.1 SAX ... 809
30.2.2 SimpleXML ... 814
30.2.3 DOM-Zugriff ... 818
30.2.4 Validierung ... 824
30.2.5 XSLT ... 826
30.3 XMLReader und XMLWriter ... 827
30.3.1 XMLReader ... 828
30.3.2 XMLWriter ... 829
30.4 EXIF ... 830
30.4.1 Vorbereitung ... 831
30.4.2 Umsetzung ... 831

31 Grafiken mit PHP 837

31.1 Vorbereitungen ... 837
31.1.1 Installation ... 837
31.2 GD 2 im Einsatz ... 838
31.2.1 Grundgerüst ... 838
31.2.2 Text ... 841
31.2.3 Formen ... 843
31.2.4 Linien und Stile ... 845
31.2.5 Ausgabe der Bilder ... 846
31.2.6 Bildbearbeitung ... 846
31.2.7 Dynamisches Diagramm ... 851
31.3 Die Alternativen ... 857
31.3.1 ImageMagick ... 857
31.3.2 GMagick ... 857
31.3.3 NetPBM ... 858

32 PDF mit PHP 859

32.1 Vorbereitung ... 860
32.1.1 FPDF ... 860
32.1.2 PDFlib ... 860
32.1.3 Haru ... 861
32.2 FPDF ... 861
32.2.1 Grundlagen ... 861
32.2.2 Zeichnen ... 863
32.3 PDFlib ... 864
32.3.1 Grundlagen ... 864
32.3.2 Zeichnen ... 867
32.3.3 Tortendiagramm mit PDFlib ... 871
32.3.4 Besonderheiten ... 874
32.4 Haru ... 875
32.5 Andere Bibliotheken ... 877
32.5.1 Bibliotheken ... 877

TEIL VII Administration und Sicherheit

33 Sicherheit 881

33.1 Benutzereingaben 883
33.2 XSS 886
33.3 SQL Injection 890
33.4 Versteckte Felder? 893
33.5 Input-Filter 896
33.6 Cross-Site Request Forgery 899
33.7 Screen Scraping und CAPTCHAs 904
33.8 Passwörter verschlüsseln 909
33.9 Fazit 913

34 Authentifizierung 915

34.1 Apache-Authentifizierung 916
34.2 IIS-Authentifizierung 920
34.3 HTTP-Authentifizierung von Hand 922
34.4 Fazit 924

35 Konfigurationsmöglichkeiten in der »php.ini« 927

35.1 Wo konfigurieren? 927
35.1.1 Speicherort 927
35.1.2 Andere Konfigurationsdateien 928
35.2 Was konfigurieren? 931
35.3 Fazit 937

36 Fehlersuche und Debugging 939

36.1 Debugging von Hand 941
36.2 Debugging mit DBG 943
36.3 Debugging mit Xdebug 947
36.4 Auflösung 951

37 Apache-Funktionen 955

37.1 Vorbereitungen 955
37.2 Anwendungsbeispiele 956
37.2.1 Informationen über Apache 956
37.2.2 HTTP-Header auslesen 957
37.2.3 URI-Informationen 959
37.2.4 Andere Servertechnologien einbinden 960
37.2.5 Apache-Prozess beenden 962

TEIL VIII PHP erweitern

38 PEAR erweitern 965

38.1 Programmieren 965
38.1.1 Die Idee 966
38.1.2 Coding Standards 969
38.1.3 Dokumentation 971
38.1.4 Paket erstellen 975
38.1.5 Der Beispielcode 982
38.2 Code hinzufügen 995
38.2.1 Projekt anmelden 996
38.2.2 Projekt zur Abstimmung stellen 998
38.2.3 Projekt hochladen 999
38.3 Pflege 1000
38.3.1 Versionsverwaltung 1000
38.3.2 Das Bug-System von PEAR 1001
38.4 Epilog 1001

39 PHP-Erweiterungen 1003

39.1 Programmieren 1004

39.2 Kompilieren 1010

39.3 Testen 1013

40 Zu PHP beitragen 1017

40.1 Patches für PHP 1017

40.1.1 Bugfix 1018

40.1.2 Neues Feature 1024

40.2 Weitere Möglichkeiten 1025

Index 1027

Index

- 118
-- 119
` 135
^ 131, 292
__FILE__ (Konstante) 731
__soapCall() (Methode) 778
! 129
!= 122
!== 122, 952
? 293
? : 135
. ..... 191, 292
... . 166, 249
... (Operator) 334
.htaccess 916, 928
.php 35
.php3 35
.php4 35
' 101, 108
(&& 128
[^ADZ] 293
[] 230, 292
[a-z] 292
{ 138
{,8} 293
{} 293
} 138
@ 134
* 118, 293
** 120
**= 120
*= 119
/ 118, 295
/* 95
/* ... */ 95
// 95
/= 119
\ 102
\" 101, 108, 109
\\ 108, 292
\\\ 109
\$ 109
\0 207
\000 109
\1, \2, \3 293
\B 292
\b 292
\D 292
\d 292
\n 108, 109, 194, 207
\r 109, 207
\S 292
\s 292
\t 109, 207
\v 207
\W 292
\w 292
\x00 109
& 113, 131
# 95
% 118
%= 119
+ 118, 293
++ 119
+= 119
< 121
<?php 93
<% 94
<< 131
<<< 184
<= 122
<> 122

 194
-= 119
= 103, 231
== 122
=== 122, 952
-> 307
> 121
>= 122
>> 131
| 131, 293
|| 128
~ 131
$ 102, 292
$_COOKIE 114, 254, 466
$_ENV 114, 254
$_FILES 114, 254, 759
$_GET 114, 254, 883
$_POST 114, 254, 883
$_REQUEST 114, 254
$_SERVER 114, 254

$_SERVER;\HTTP_AUTHORIZATION\"" 924
$_SERVER;\PHP_AUTH_PW\"" 923
$_SERVER;\PHP_AUTH_USER\"" 923
$_SESSION 114, 254, 486
$GLOBALS 114, 170
$HTTP_COOKIE_VARS 400, 467
$HTTP_GET_VARS 114, 400
$HTTP_POST_FILES 400
$HTTP_POST_VARS 400
$HTTP_SERVER_VARS 253, 400
$this 312
10gen 697

A

abs() (Funktion) 267
abstract 332
Abstrakte Klassen 331
Abstraktionsklassen 547
ACID-Prinzip 543
Acrobat 859
Active Server Pages 94
add() (Methode) 289
addcslashes() (Funktion) 215
addFunction() (Methode) 777
Addition 117
AddPage() (Methode) 862
addPage() (Methode) 875
addslashes() (Funktion) 214, 552, 790
Adobe 859
Ajax 794
Aliase 202
allow_url_fopen (php.ini) 99
American Standard Code for Information Interchange 123
Anbieter 762
and 128
Anführungszeichen 101, 108
Anonyme Funktionen 174
Anweisungen 95
Anweisungsblock 94, 138
Apache 32, 45, 955
2.x 51
andere Technologien einbinden 960
Authentifizierung 916
htdocs 52
Installation 66
Prozess beenden 962
Version 2 955
Version 2.2 955
virtueller Pfad 961
Apache für Windows 51
apache_get_modules() (Funktion) 956
apache_get_version() (Funktion) 956
Apache-Lizenz 972
appendChild() (Methode) 821
Architekturen (von Software) 762
Arithmetische Operatoren 117
Kurzformen 118
array_combine() (Funktion) 246
array_fill() (Funktion) 248
array_filter() (Funktion) 198
array_flip() (Funktion) 251
array_key_exists() (Funktion) 250, 469
array_keys() (Funktion) 250
array_map() (Funktion) 240
array_merge_recursive() (Funktion) 246
array_merge() (Funktion) 244
array_pop() (Funktion) 241
array_push() (Funktion) 242
array_rand() (Funktion) 251
array_reverse() (Funktion) 251
array_search() (Funktion) 250
array_shift() (Funktion) 242
array_slice() (Funktion) 242
array_splice() (Funktion) 244
array_unshift() (Funktion) 242
array_values() (Funktion) 250
array_walk_recursive() (Funktion) 239
array_walk() (Funktion) 239
array() 229
Arrays 103, 229
ändern 231
assoziative 233
durchsuchen 250
Elemente löschen 232
erstellen 229
globale 252
Iteration 234
Kurzschreibweise 233
mit Schleifen 234
multidimensionale 234
sortieren 251
superglobale 114, 252
transformieren 241
untersuchen 240
Variablen 246
verbinden 244
ASCII 123
Code 123
Umwandlung 225
ASP 94

asp_tags (php.ini) 94
Assoziative Arrays 233
Assoziativität 137
asXML() (Methode) 817
Atomarität → Atomicity
Atomicity 543
attributes() (Methode) 816
Ausdruck 95
Aussprachetest 224
Authentifizierung
Apache 916
Basic 916
Digest 916
HTTP 915, 922
IIS 920
mit Sessions 493
Schlussfolgerungen 924
Sessions 915
Webserver 915
AuthName 918
AuthType 918
autoconf 66
Ayala, Dietrich 767

B

Backslash 108
base_convert 263
base64_decode() (Funktion) 750
base64_encode() (Funktion) 521, 732, 750
BC (Binary Calculator) 268
bcadd() (Funktion) 269
bcdiv() (Funktion) 269
bcmath.scale (php.ini) 268
bcmod() (Funktion) 269
bcmul() (Funktion) 269
bcomp() (Funktion) 269
bcpow() (Funktion) 269
bcpowmod() (Funktion) 270
bcscale() (Funktion) 268
bcsqrt() (Funktion) 270
bcsub() (Funktion) 269
Bedingung 138
Bildbearbeitung 846
Bild verkleinern 847
Farbkorrektur 853
Graustufen 848
Binäres System 130, 132, 262
Binärmodus 722
Binary Large Objects 660
bindec() (Funktion) 262
bison 66
Bitweise Operatoren 129
BLOB 660
BLOB (SQL) 532
Bogenmaß 267
bool → boolean
boolean 103
BOOLEAN (SQL) 532
Boutell, Thomas 838
break 145, 146, 156
Breakpoint 946
Browserweiche 255
BSD-Lizenz 972
Bundesamt für Sicherheit in der Informationstechnik 910
By Ref 113
bzip2 753

C

C 31, 1003
Compiler 66
call_user_func_array() (Funktion) 181
call_user_func() (Funktion) 180
call() (Methode) 771
Camel Case 106, 975
CAPTCHA 904, 967
Caraveo, Shane 767
case 145
Case-insensitiv 204
ceil 267
ceil() (Funktion) 267
Cell() (Methode) 862
CGI 42, 955
Changelog 39
CHAR (SQL) 532
char() (Funktion) 203
Checkboxen 406
validieren 418
checkdate() (Funktion) 281, 288
children() (Methode) 816
chop() (Funtkion) 207
chr() (Funktion) 225
chunk_split() (Funktion) 192
class 307
CLI 43
Client-Server-Modell 34
Coding Standard 969
Trennung Code und Inhalt 96
colorat() (Funktion) 850
colorsetpixel() (Funktion) 850

Comment Spam ... 966
COMMIT (SQL) ... 544
Common Gateway Interface → CGI
compact() (Funktion) ... 248
compress.bzip2:// ... 743
compress.zlib:// ... 743
configureWSDL() (Methode) ... 772
Consistency ... 543
Content Security Policy ... 890
Content-type ... 838
continue ... 158
convert.base64-decode (Filter) ... 750
convert.base64-encode (Filter) ... 750
convert.quoted-printable-decode (Filter) ... 751
convert.quoted-printable-encode (Filter) ... 751
Cookies ... 455, 481
Ablaufdatum ... 461
auslesen ... 466
Browser ... 456
Cookie-Test ... 475
Domain ... 462
Einschränkungen ... 458
HTTPS ... 463
löschen ... 469
neue Cookies ... 473
permanent ... 461
persistent ... 461
Pfad ... 462
Probleme ... 477
setzen ... 460
Sonderzeichen ... 466
Spezifikation ... 457
temporär ... 461
copy() (Funktion) ... 729
CORBA ... 761
cos() (Funktion) ... 267
COUNT (SQL) ... 541
COUNT_RECURSIVE (Konstante) ... 240
count() (Funktion) ... 234
Countdown ... 286
CPDF ... 877
CREATE DATABASE (SQL) ... 533
CREATE TABLE (SQL) ... 533
create_function() (Funktion) ... 180
createFromFormat() (Methode) ... 279
createTextNode() (Methode) ... 821
Cross-Site Request Forgery → CSRF
Cross-Site Scripting ... 886
crypt() (Funktion) ... 226, 911
CSP → Content Security Policy
CSRF ... 562
CSV ... 367
current() (Funktion) ... 238
current() (Methode) ... 355

D

Data Source Name ... 550
DATE (SQL) ... 532
date_default_timezone_get() (Funktion) ... 285
date_default_timezone_set() (Funktion) ... 285
date_parse_from_format() (Funktion) ... 280
date() (Funktion) ... 270, 272, 596
Dateibrowser ... 737
Dateien ... 719
auslesen ... 725
Einstellungen ... 739
Handling ... 720
Infos ... 727
Modi ... 721
Operationen ... 728
Pfade ... 723
Rolle im Dateisystem ... 726
schreiben ... 723
sperren ... 734
Dateioperationen ... 719
Datenbanken
Abstraktionsklassen ... 547
Aggregatfunktionen ... 541
Datenbankdesign ... 538
Datentypen ... 531
Datumswerte ... 532
Joins ... 540
MS SQL Server ... 625
MySQL ... 567
mysqli ... 567
Oracle ... 647
PostgreSQL ... 671
Sequenzen ... 659
Sessions speichern ... 495
SQLite ... 603
Stored Procedures ... 544, 635
Tabellen ... 530
Transaktionen ... 543, 586
Datentypen
Arrays ... 103
BLOB ... 660
Boolean ... 103

Datentypen (Forts.)
Double ... 103
feststellen ... 104
für Datenbanken ... 531
Integer ... 103
lo ... 684
LOB ... 660
MySQL ... 579
NULL ... 103
Object ... 103
Real ... 103
Resource ... 103
SERIAL ... 681
String ... 103
DateTime ... 283
DateTime (Klasse) ... 270
DateTime (Objekt) ... 275
Datum ... 259, 270
Ablauf für Cookies ... 461
aktuelles ... 270
aus Strings ... 279
beliebige Werte ... 276
Countdown ... 286
deutsch ... 282
Formatangaben ... 272
formatieren ... 282
GNU-Richtlinien ... 281
lokale Einstellung ... 285
Zeitstempel ... 277
Dauerhaftigkeit → Durability
DB
Verbindung zu Datenbanken ... 550
DBG ... 943
DCOM ... 761
Debugger
Breakpoint ... 946
Xdebug ... 947
Debugging ... 939
DBG ... 943
von Hand ... 941
decbin() (Funktion) ... 262
dechex() (Funktion) ... 262
DECIMAL (SQL) ... 532
decoct() (Funktion) ... 262
default ... 147
define() (Funktion) ... 114
deg2rad() (Funktion) ... 267
Deklarieren ... 163
Dekrement ... 119
Destruktor ... 321
Diagramme
dynamisch ... 851
mit PDF ... 871
Torte ... 844
diff() (Methode) ... 289
dir (Klasse) ... 730
Direktive ... 43
dirname() (Funktion) ... 731
--disable-mysql (Schalter) ... 573
--disable-session (Schalter) ... 482
display_errors (php.ini) ... 96
Dithering ... 850
Division ... 117
DLL ... 57
DocBook ... 971
documentElement (Eigenschaft) ... 821
DOM ... 808, 818
XPath ... 823
dom_import_simplexml() (Funktion) ... 818
DOMDocument (Klasse) ... 818
DomElement (Klasse) ... 824
DomNode (Klasse) ... 820, 824
DomXPath (Klasse) ... 823
Double ... 103
do-while ... 160
DSN ... 550
DTD ... 805, 825
Durability ... 544

E

each() (Funktion) ... 239
echo (Anweisung) ... 96, 100, 184, 941
Eigenschaften ... 306, 308
Namen und Werte ... 354
Eindeutige ID ... 226
else ... 141
elseif ... 139
E-Mails ... 507
Bcc ... 515
Cc ... 515
Content-ID ... 523
Einstellungen ... 511
IMAP ... 525
laden ... 525
MIME ... 517
POP ... 525
Prioritäten ... 516
Reply-To ... 515
versenden ... 512
X-Priority ... 515

empty() (Anweisung) ... 110
empty() (Funktion) ... 240, 469
--enable-bcmath (Schalter) ... 268
--enable-exif (Schalter) ... 831
--enable-ftp (Schalter) ... 755
--enable-soap (Schalter) ... 768
--enable-trans-sid (Schalter) ... 484
Encapsulation ... 321
end of file → s, eof
end() (Funktion) ... 239
endfor ... 152
Endlosschleifen ... 151
endwhile ... 156
ENT_COMPAT (Konstante) ... 218
ENT_NOQUOTES (Konstante) ... 218
ENT_QUOTES (Konstante) ... 218
Entwerten ... 214
Entwurfsmuster ... 324, 379
  Factory ... 389
  Gang of Four ... 379
  MVC ... 382
  Singleton ... 387
eof ... 725
ereg_replace() (Funktion) ... 295
ereg() (Funktion) ... 294
eregi_replace() (Funktion) ... 295
eregi() (Funktion) ... 295
error_reporting (php.ini) ... 96
error.log ... 88
Erweiterungen ... 56
  kompilieren ... 1010
  Modul nicht gefunden ... 83
  PEAR ... 965
  PECL ... 1003
  php.ini ... 937
escape() (JavaScript) ... 793
Escape-Sequenzen ... 108
Event-Handler ... 810
Exchangeable Image File Format → EXIF
exec() (Funktion) ... 729
EXIF ... 830
explode() (Funktion) ... 197, 198
Exponential-Operator ... 120
export() (Methode) ... 357
Expression → Ausdruck
eXtensible Markup Language ... 803
eXtensible Stylesheet Language Transformation ... 806
Externes Skript ... 96
EXTR_IF_EXISTS (Konstante) ... 248
EXTR_OVERWRITE (Konstante) ... 247
EXTR_SKIP (Konstante) ... 247
extract() (Funktion) ... 246

F

Factory ... 389
Fallunterscheidungen ... 138
  alternative Formen für if ... 143
  if ... 138
  Kurzformen für if ... 142
  switch ... 145
  Vergleich if und switch ... 149
  verschachtelt ... 144
FastCGI ... 42
fclose() (Funktion) ... 722
FDF ... 877
Fehler ... 939
  -meldung ... 939
  Parser ... 939
  -suche ... 940
Fehlerunterdrückung ... 134
feof() (Funktion) ... 725
fgetc() (Funktion) ... 726
fgets() (Funktion) ... 725, 748
FI → Forms Interpreter
Fielding, R. T. ... 764
FILE_APPEND (Konstante) ... 724
file_exists() Funktion ... 727
file_get_contents() (Funktion) ... 684, 725, 744, 783, 810
File_PDF ... 860
file_put_contents() (Funktion) ... 724, 988
file:// ... 742
file() (Funktion) ... 735
Filemon ... 720
Files → Dateien
Filter ... 749
filter ... 743
filter_input() (Funktion) ... 896
filter_var() (Funktion) ... 898
flex ... 66
flock() (Funktion) ... 735
floor() (Funktion) ... 267
Fonts ... 842
  GD 2 ... 841
  True Type ... 842
fopen() (Funktion) ... 721, 742
for ... 149
  alternative Syntax ... 152
  für Arrays ... 234

foreach
  für Arrays ... 235
  Index auslesen ... 237
Forms Date Format ... 877
Forms Interpreter ... 31
Formulare
  Auswahlliste ... 395
  Checkbox ... 395
  einzeiliges Passwortfeld ... 395
  einzeiliges Textfeld ... 395
  mehrzeiliges Textfeld ... 395
  Radiobutton ... 395
  register_long_arrays ... 400
  Versendeschaltfläche ... 395
Fowler, Martin ... 380
fpassthru() (Funktion) ... 725
FPDF ... 860, 861
FPDF (Klasse) ... 861
Free PDF → FPDF
fseek() (Funktion) ... 726
fsockopen() (Funktion) ... 747
FTP ... 741
FTP_ASCII (Konstante) ... 756
FTP_BINARY (Konstante) ... 756
ftp:// ... 743
ftps:// ... 743
FTP-Streams ... 749
func_get_arg() (Funktion) ... 168
func_get_args() (Funktion) ... 168
func_num_args() (Funktion) ... 167
function (Schlüsselwort) ... 163
function_exists() (Funktion) ... 179
Funktionen ... 163
  Aliase ... 202
  anonyme ... 174
  deklarieren ... 163
  Gültigkeit von Variablen ... 168
  nicht definiert ... 87
  Parameter ... 164
  rekursive ... 175
  Rückgabewert ... 170
  Variable ... 173
  von PHP ... 181
  Vorgabewert ... 165
fwrite() (Funktion) ... 723, 748

G

Gamma, Erich ... 390
Gang of Four ... 379
Gästebuch ... 557, 732
gcc ... 66
GD 2 ... 837
GDChart ... 853
Generatoren ... 177
get_html_translation_table() (Funktion) ... 219
getdate() (Funktion SQL) ... 545
getdate() (Funktion) ... 270
getElementById() (Methode) ... 820
getElementsByTagname() (Methode) ... 820
getFont() (Methode) ... 875
getIterator() (Methode) ... 355
getrandmax() (Funktion) ... 264
getStreamSize() (Methode) ... 875
getTimestamp() (Methode) ... 277
gettype() (Funktion) ... 104
GIF ... 838
Git ... 1017
GitHub ... 1017, 1020
Gleichheit ... 122
  genaue ... 122
global (Schlüsselwort) ... 169
Globale Variablen ... 168
gmdate() (Funktion) ... 286
gmmktime() (Funktion) ... 286
gmstrftime() (Funktion) ... 286
GMT ... 286
Google ... 966
go-pear.phar ... 72
goto ... 161
GPL ... 972
Grafiken ... 837
  Bild verkleinern ... 847
  Dateityp ... 838
  Formen ... 843
  Graustufen ... 848
  Hintergrundfarbe ... 840
  Koordinaten ... 840
  Text ... 841
Greenwich Mean Time → GMT
Gutmans, Andi ... 31
gzip ... 753

H

handle() (Methode) ... 777
Haru ... 875
HaruDoc (Objekt) ... 875
HaruPage (Objekt) ... 875
Hash ... 225
hash() (Funktion) ... 226
Header, auslesen ... 957

header() (Funktion) 473, 865, 922
heredoc-Syntax 184
Hexadezimales System 109, 262
hexdec() (Funktion) 262
Hilfsquellen 89
Holzgraefe, Hartmut 1004
Hosting 39
htdocs 52, 71
HTML 216
Einbau von PHP 93
Sonderzeichen 217
html_entity_decode() (Funktion) 219
HTML2PDF 877
htmlentities() (Funktion) 217, 219
htmlspecialchars() (Funktion) 219, 402, 472, 552, 596, 725, 756, 810, 889
htpasswd 917
HTTP 33, 455, 741
Authentifizierung 915, 922
Header auslesen 957
Set-Cookie2 473
X-Mailer 514
http:// 743
httpd.conf 70, 85, 928
HTTPS 463, 741
https:// 743
HTTP-Streams 744
Hypertext Preprocessor 31
HyperText Transfer Protocol 33

I

idate() (Funktion) 285
if 138
Kurzformen 142
mit Doppelpunkt 143
verschachtelt 144
IIS 45, 47
Authentifizierung 920
Image_CAPTCHA 982
imagecolorallocate() (Funktion) 840
imagecolorallocatealpha() (Funktion) 849
imagecolorclosest() (Funktion) 849
imagecolorclosestalpha() (Funktion) 849
imagecolorsforimage() (Funktion) 850
imagecolorsforindex() (Funktion) 848
imagecolorstotal() (Funktion) 848
imagecolortransparent() (Funktion) 849
imagecopyresized() (Funktion) 847
imagecreate() (Funktion) 839
imagecreatefromGIF() (Funktion) 846
imagecreatefromJPEG() (Funktion) 846
imagecreatefromPNG() (Funktion) 846
imagecreatefrompng() (Funktion) 846
imagecreatetruecolor() (Funktion) 839
imagedestroy() (Funktion) 840
imageellipse() (Funktion) 843, 845
imagefilledarc() (Funktion) 843
imagefilledellipse() (Funktion) 843
imagefilledpolygon() (Funktion) 843
imagefilledrectangle() (Funktion) 840, 843
imagegif() (Funktion) 839, 846
imagejpeg() (Funktion) 839, 846
imageline() (Funktion) 845
imagelinestyle() (Funktion) 845
imageloadfont() (Funktion) 842
ImageMagick 837, 857
imagepng() (Funktion) 839, 846
imagerectangle() (Funktion) 845
imagesetbrush() (Funktion) 846
imagesetthickness() (Funktion) 845
imagestring() (Funktion) 841
imagestringup() (Funktion) 841
imagetruecolortopalette() (Funktion) 850
imagettfbbox() (Funktion) 842
imagettftext() (Funktion) 842
IMAP 525
IMG_COLOR_STYLED (Konstante) 845
IMG_COLOR_STYLEDBRUSHED (Konstante) 846
implode() (Funktion) 202
importStylesheet() (Methode) 826
in_array() (Funktion) 241
include_once() (Anweisung) 98
include_path (php.ini) 99, 723
include() (Anweisung) 96
ini_set() (Funktion) 100, 929
Inkrement 119
input 743
Input-Filter 896
Installation 41
Distribution 65
häufige Fehler 79
Linux 64
Mac 58
MS SQL Server 625
MySQL 568
Oracle 647
Pakete 65
PDF 860

Installation (Forts.)
PEAR 72
PostgreSQL 671
SQLite 604
testen 51
von Hand 47, 66
Web-Service-Pakete 768
Windows 44
XML 809
INT (SQL) 532
integer 103
Internet Explorer
Cookies 456
Fehlermeldungen 79
Internet Information Services → IIS
Interpretieren 34
IP
Identifizierung 920
prüfen 504
is_array() (Funktion) 241
is_bool() (Funktion) 104
is_callable() (Funktion) 174
is_dir() (Funktion) 727, 737
is_executable() (Funktion) 727
is_file() (Funktion) 727
is_link() (Funktion) 727
is_null() (Funktion) 112
is_numeric() (Funktion) 104
is_readable() (Funktion) 727
is_string() (Funktion) 104
is_uploaded_file() (Funktion) 727, 758
is_writable() (Funktion) 727
ISO 803
ISO-8859-1 812
ISO-8859-15 218
Isolation 544
isset() (Anweisung) 109
isset() (Funktion) 179, 240, 402, 469
Iteration 355
für Arrays 234
per Funktion 238

J

JavaScript 787, 888
Ausführreihenfolge 788
JavaScript Object Notation 781
join() (Funktion) 202
Joins 540
Jones, Christopher 1025
Joomla! 33
JPEG 838
JSON 781, 798
json_decode() (Funktion) 782, 798
json_encode() (Funktion) 782, 798

K

Kalender 281
Kanonisierung 737
Kapselung 321
KCachegrind 950
kdesdk 950
Kerberos 920
key() (Funktion) 239
key() (Methode) 355
Klassen 305
abstrakte 331
-typen 347
Knuth, Donald 224
Kommentare 95
Kompressions-Streams 753
Komprimierung 758
Konditionaler Operator 135
Konfiguration 931
Konkatenation 121, 191
Konsistenz → Consistency
Konstanten 114, 333
magische 345
mathematische 260
Konstruktor 320
Konsument 762
Kontrollkästchen → Checkboxen
Konvertierung 261
Koordinaten 840
PDFlib 865
ksort() (Funktion) 252

L

LAMP 44
LAMPP 65
Large Objects 660
Late Static Binding 305, 330
Lerdorf, Rasmus 31, 896, 1004
Lesezugriff 721
levenshtein() (Funktion) 224
libbcmath 268
libHaru 875
libtool 66
libxml2 809
LIKE (SQL) 536

Linux 64
Installationspakete 65
RPM 43
list() (Sprachkonstrukt) 246
LiveScript 787
Lizenzen 972
GPL 972
PHP-Lizenz 972
load() (Methode) 660, 819
loadHTML() (Methode) 819
loadHTMLFile() (Methode) 819
loadXML() (Methode) 819
LOB 660
localhost 52
location.href (JavaScript) 793
LOCK_EX (Konstante) 724
Logging 933
Logische Operatoren 128
ltrim() (Funktion) 207
Lynx 756
LZW 838

M

M_1_PI (Konstante) 261
M_2_PI (Konstante) 261
M_2_SQRTPI (Konstante) 261
M_E (Konstante) 261
M_LN10 (Konstante) 261
M_LN2 (Konstante) 261
M_LOG10E (Konstante) 261
M_LOG2E (Konstante) 261
M_PI (Konstante) 261
M_PI_2 (Konstante) 261
M_PI_4 (Konstante) 261
M_SQRT1_2 (Konstante) 261
M_SQRT2 (Konstante) 261
Mac OS X 58
Mac, Distributionen 62
Magische Konstante 345
mail.force_extra_parameters (php.ini) 511
mail() (Funktion) 512
Mailinglisten 89
MariaDB 568
Maschine-Maschine-Kommunikation 762
Mathematik 259
Bogenmaß 267
Funktionen 259
höhere Genauigkeit 267
Konstanten 260
Mathematik (Forts.)
konvertieren 261
Maximalzahl 265
Minimalzahl 265
Operatoren 259
Potenz 259
Quadratwurzel 259
runden 266
Zufallszahlen 264
max() (Funktion) 265
mcrypt 226
MD5 911
md5_file() (Funktion) 225, 978
md5() (Funktion) 225, 911
Merz, Thomas 864
Methoden 306, 310
Direktzugriff 313
Parameter 310
Rückgabewert 311
Microsoft 625
Microsoft SMTP Service 510
Microsoft SQL Server 625
Microsoft SQL Server Driver for PHP 627
Microsoft SQL Server Express Edition 625
microtime() (Funktion) 277
Migration, Objektorientierung 369
MIME 517, 959
Content-Type 518
min 265
min() (Funktion) 265
mkdir() (Funktion) 729
mktime() (Funktion) 278, 461
modify() (Methode) 289
Modular programmieren 96
Modulo 118, 158
MongoClient (Klasse) 702
MongoDB 697
Daten abfragen 704
Daten aktualisieren 706
Daten einfügen 703
Daten löschen 707
Download 697
Einstellungen 715
Installation 698
PHP-Extension 699
Verbindungsaufbau 702
MongoDB Inc. 697
MS SQL Server
Daten ausgeben 639
Daten bearbeiten 643

MS SQL Server (Forts.)
Daten eintragen 638
Daten löschen 641
letzter Autowert 634
mssqli.max_links (php.ini) 602
mt_getmaxrand() (Funktion) 265
mt_rand() (Funktion) 251, 265
mt_srand() (Funktion) 265
Multidimensionale Arrays 234
Multiplikation 117
MVC 382
MySQL 567
alte Versionen 589
Autowert 576
Besonderheiten 585
Daten ausgeben 595
Daten bearbeiten 599
Daten löschen 597
Datenbankzugriff 574
Datentypen 579
Fehlerbehandlung 588
Installation 568
letzter Autowert 585
Rückgabewerte 580
TIMESTAMP 593
Transaktionen 586
Verbindungsaufbau 575
mysql_connect() (Funktion) 590
mysql_errno() (Funktion) 590
mysql_error() (Funktion) 590
mysql_fetch_array() (Funktion) 591
mysql_fetch_assoc() (Funktion) 591
mysql_fetch_object() (Funktion) 591
mysql_fetch_row() (Funktion) 591
mysql_insert_id() (Funktion) 592
mysql_query() (Funktion) 590
mysql_real_escape_string() (Funktion) 591
mysql_select_db() (Funktion) 590
mysqli 567, 892
autocommit() (Methode) 587
bind_param() (Methode) 578
close() (Methode) 577
commit() (Methode) 587
errno (Eigenschaft) 588
error (Eigenschaft) 588
execute() (Methode) 579
insert_id (Eigenschaft) 585
objektorientiert 576
query() (Methode) 580
real_escape_string() (Methode) 578
rollback() (Methode) 587
MYSQLI_ASSOC (Konstante) 585
mysqli_autocommit() (Funktion) 587
MYSQLI_BOTH (Konstante) 585
mysqli_close() (Funktion) 575
mysqli_commit() (Funktion) 587
mysqli_connect_errno() (Funktion) 588
mysqli_connect_error() (Funktion) 588
mysqli_connect() (Funktion) 575
mysqli_errno() (Funktion) 588
mysqli_error() (Funktion) 588
mysqli_fetch_array() (Funktion) 584
mysqli_fetch_assoc() (Funktion) 581, 584
mysqli_fetch_fields() (Funktion) 585
mysqli_fetch_object() (Funktion) 582
mysqli_fetch_row() (Funktion) 583
mysqli_insert_id() (Funktion) 585
MYSQLI_NUM (Konstante) 585
mysqli_prepare() (Funktion) 578, 588
mysqli_query() (Funktion) 576, 580
mysqli_real_escape_string() (Funktion) 578, 891
mysqli_result
fetch_array() (Methode) 584
fetch_assoc() (Methode) 581
fetch_fields() (Methode) 585
fetch_object() (Methode) 582
fetch_row() (Methode) 583
mysqli_result (Klasse) 581
mysqli_rollback() (Funktion) 587
mysqli_select_db() (Funktion) 575
mysqli_stmt_bind_param() (Funktion) 578, 579
mysqli_stmt_errno() (Funktion) 588
mysqli_stmt_error() (Funktion) 588
mysqli_stmt_execute() (Funktion) 579
mysqli:\:close() (Methode) 576
mysqli.default_host (php.ini) 602
mysqli.default_port (php.ini) 602
mysqli.default_pw (php.ini) 602
mysqli.default_socket (php.ini) 602
mysqli.default_user (php.ini) 602

N

Namensräume 806
namespace (Schlüsselwort) 373
Namespaces 371
alias 376
anlegen 373
ermitteln 375
verwenden 374

natsort() (Funktion) ... 251
NDATA ... 812
Netscape ... 787
new ... 307
next() (Funktion) ... 238
next() (Methode) ... 355
nl2br (Funktion) ... 194
NoSQL ... 697
nowdoc-Syntax ... 185
NULL ... 103, 165, 179
nuSOAP ... 767, 768
Client ... 771
Server ... 769
WSDL ... 772
nusoap_server() (Methode) ... 769
Nutzerauthentifizierung mit Sessions ... 493
Nutzersitzung → Sessions

O

OASIS ... 764
Objekte ... 103, 305
serialisieren ... 350
vergleichen ... 349
Objektmodelle ... 324
Objektorientierte Architektur ... 762
Objektorientierung ... 305
$this ... 312
abstrakte Klassen ... 331
Iteration ... 355
Migration von PHP 4 auf PHP 5 ... 369
überladen ... 314, 333
überschreiben ... 317
Unterschiede zwischen PHP 4 und PHP 5 ... 368
Vererbung ... 315
Zugriffskontrolle ... 321
OCI_ASSOC (Konstante) ... 658
oci_bind_by_name() (Funktion) ... 655
oci_close() (Funktion) ... 652
OCI_COMMIT_ON_SUCCESS (Konstante) ... 653
oci_commit() (Funktion) ... 653
oci_connect() (Funktion) ... 652
OCI_DEFAULT (Konstante) ... 653
oci_error() (Funktion) ... 653
oci_execute() (Funktion) ... 653
oci_fetch_all() (Funktion) ... 658
oci_fetch_assoc() (Funktion) ... 657
oci_fetch_object() (Funktion) ... 657
oci_fetch() (Funktion) ... 652, 656
oci_new_descriptor() (Funktion) ... 660
OCI_NUM (Konstante) ... 658
oci_parse() (Funktion) ... 653, 656
oci_result() (Funktion) ... 656
ocifetch() (Funktion) ... 652
ocilogon() (Funktion) ... 652
octdec() (Funktion) ... 262
OIDS ... 678
Oktales System ... 109, 262
Onlinehandbuch ... 89
OOA ... 762
OOP → Objektorientierung
Open Web Application Security Project → OWASP
opendir() (Funktion) ... 730
OpenSSL ... 741
Operanden ... 117
Operationen
arithmetische ... 117
Operator
binär ... 117
ternär ... 117
unär ... 117
Operatoren ... 117
bitweise ... 129
Dekrement ... 119
Inkrement ... 119
konditionaler ... 135
logische ... 128
Präferenz ... 136
Rangfolge ... 136
Vergleichs- ... 121
or ... 128
Oracle ... 647
Besonderheiten ... 659
BLOB ... 660
Daten ausgeben ... 664
Daten bearbeiten ... 667
Daten eintragen ... 662
Daten löschen ... 665
DUAL ... 654
Fehlerbehandlung ... 653
Installation ... 647
letzter Autowert ... 659
LOB ... 660
Rückgabewerte ... 656
Sequenzen ... 659
Umgebungsvariablen ... 649
Verbindungsaufbau ... 652
ord() (Funktion) ... 203, 225

ORDER BY (SQL) ... 535
output ... 743
output_buffering (php.ini) ... 464
OWASP ... 881

P

PageRank ... 966
Panda ... 877
Parameter ... 164
flexible Anzahl ... 166
für Methoden ... 310
Vorgabewert ... 165
parse_url() (Funktion) ... 222
Parser ... 808
Pascal ... 106
Pascal-Case ... 106
Password Hashing API ... 911
password_get_info() (Funktion) ... 911
password_hash() (Funktion) ... 911
password_needs_rehash() (Funktion) ... 911
password_verify() (Funktion) ... 911
Passwortschutz ... 493
PDF ... 859
andere Bibliotheken ... 877
Bibliotheken ... 860
Installation ... 860
pdf_begin_page() (Funktion) ... 864
pdf_fill() (Funktion) ... 867
pdf_findfont() (Funktion) ... 865
pdf_new() (Funktion) ... 864
pdf_open_file() (Funktion) ... 864
pdf_rect() (Funktion) ... 867
pdf_restore() (Funktion) ... 868
pdf_set_font() (Funktion, veraltet) ... 865
pdf_set_info() (Funktion) ... 864
pdf_setcolor() (Funktion) ... 867
pdf_setfont() (Funktion) ... 865
pdf_show_xy() (Funktion) ... 865
pdf_translate() (Funktion) ... 867
PDFlib ... 860, 864
Besonderheiten ... 874
PDO
beginTransaction() (Methode) ... 556
commit() (Methode) ... 556
exec() (Methode) ... 551
PHP Data Objects ... 548
prepare() (Methode) ... 552
query() (Methode) ... 555
rollBack() (Methode) ... 556
PDOStatement
bindParam() (Methode) ... 552
execute() (Methode) ... 552
fetch() (Methode) ... 554
PEAR ... 72
Bugsystem ... 1001
Channels ... 74
Coding Standards ... 969
Dokumentation ... 971
Entwicklerkonto ... 995
erweitern ... 965
File_PDF ... 860
Handbuch ... 77
Installation ... 72
Lizenz ... 972
Mailinglisten ... 965
ohne Installation ... 77
Paketpflege ... 1000
Projekt anmelden ... 996
Projekt hochladen ... 999
Pyrus ... 1006
Regeln ... 965
SVN ... 1000
Text_CAPTCHA ... 906
Text_Password ... 983
UDDI ... 784
XML_SVG ... 74
PEAR::SOAP ... 767
PEAR2 ... 75
PECL ... 548, 699, 1003
PECL_Gen ... 1009
Xdebug ... 948
Performance
Sessions ... 485
Strings ... 199
Perl ... 291
reguläre Ausdrücke ... 295
Personal Homepage Tools ... 31
Pfade ... 723
tatsächlich ... 731
virtuell ... 731
pg_close() (Funktion) ... 675
pg_connect() (Funktion) ... 675
pg_escape_string() (Funktion) ... 677, 891
pg_fetch_all() (Funktion) ... 679
pg_fetch_assoc() (Funktion) ... 678
pg_fetch_object() (Funktion) ... 679
pg_fetch_row() (Funktion) ... 679
pg_insert() (Funktion) ... 681
pg_last_error() (Funktion) ... 676

pg_last_oid() (Funktion) ... 681
pg_lo_close() (Funktion) ... 684
pg_lo_create() (Funktion) ... 684
pg_lo_open() (Funktion) ... 684
pg_lo_read_all() (Funktion) ... 686
pg_lo_read() (Funktion) ... 686
pg_lo_write() (Funktion) ... 684
pg_pconnect() (Funktion) ... 675
pg_query() (Funktion) ... 676, 678
pgAdmin ... 673
PHP ... 31
Anführungszeichen ... 101
Anweisungen ... 95
Bug-Tracker ... 1018
Datentypen ... 103
erweitern ... 1003
Erweiterungen installieren ... 56
externes Skript ... 96
Funktionen ... 181
Geschichte ... 31
Grundlagen ... 93
Hilfsquellen ... 89
Homepage ... 39
in HTML ... 93
Installation ... 41
Interpreter ... 34
kompilieren ... 53
Konfigurationsdateien ... 928
Konstanten ... 114
Konzept ... 33
Logging ... 933
Mailinglisten ... 89
Onlinehandbuch ... 89
Operatoren ... 117
Patches ... 1017
Praxisprojekte ... 33
short-circuit evaluation ... 129
Verbreitung ... 33
Versionen ... 31
Versionswechsel ... 50
PHP 5.1
GD-Neuerungen ... 851
in Strings auf Zeichen zugreifen ... 192
preg_replace() ... 299
substr_compare() ... 223
substr_count() ... 210
Zeitzone ... 285
PHP 5.3 ... 367
tenärer Operator ... 136
PHP Data Objects ... 548
php_admin_flag (php.ini) ... 929
php_admin_value (php.ini) ... 929
php_flag (php.ini) ... 929
PHP_INI_ALL (php.ini) ... 930
PHP_INI_PERDIR (php.ini) ... 930
PHP_INI_SYSTEM (php.ini) ... 930
PHP_INI_USER (php.ini) ... 930
PHP_SELF ... 255
php_user_filter (Klasse) ... 751
php_value (php.ini) ... 929
php://filter ... 743, 749
php://input ... 743
php://output ... 743
php://stderr ... 743
php://stdin ... 743
php://stdout ... 743
php.exe ... 48
php.ini ... 43, 48
asp_tags ... 94
BZ2 ... 754
Direktive ... 43
Erweiterungen ... 56, 937
EXIF ... 831
File-Uploads ... 936
ini_set() ... 929
Logging ... 933
mail function ... 511
mail.force_extra_parameters ... 511
Mathematik ... 268
MySQL ... 602
register_long_arrays ... 400
Resource Limits ... 933
sendmail_from ... 511
sendmail_path ... 511
session.auto_start ... 485
session.cookie_httponly ... 505
session.gc_maxlifetime ... 486
session.name ... 491
session.referer_check ... 505
session.save_path ... 482
session.use_trans_sid ... 484
short_open_tag ... 94
SMTP ... 511
url_rewriter.tags ... 492
php.ini-development ... 48
php.ini-recommended ... 48
php5.dll ... 48
php-cgi.exe ... 48
php-cgi.ini ... 928
php-cli.ini ... 928

phpDocumentor ... 975
PHPEdit ... 945
phpinfo() (Funktion) ... 52, 1019
PHP-Lizenz ... 972
phpMyAdmin ... 571
PHPRC ... 48
PHP-SOAP ... 767, 776
Client ... 778
Server ... 777
WSDL ... 779
PHP-Streams ... 749
phpUDDI ... 784
PNG ... 839
Pool, Martin ... 251
POP ... 525
Port ... 45
Portable Document Format ... 859
Portable Operating System Interface → POSIX
pos() (Funktion) ... 239
POSIX ... 291, 293
postfix ... 509
PostgreSQL ... 671
Administration ... 673
Besonderheiten ... 680
Daten ausgeben ... 690
Daten bearbeiten ... 693
Daten eintragen ... 688
Daten löschen ... 691
Installation ... 671
letzter Autowert ... 681
lo ... 684
Rückgabewert ... 678
Verbindungsaufbau ... 675
Potenz ... 259
pow() (Funktion) ... 259
Präferenz ... 136
preg_grep() (Funktion) ... 299
preg_match_all() (Funktion) ... 297
preg_match() (Funktion) ... 295
PREG_OFFSET_CAPTURE (Konstante) 296, 297
PREG_PATTERN_ORDER (Konstante) ... 297
preg_quote() (Funktion) ... 300
preg_replace_callback() (Funktion) ... 299
preg_replace() (Funktion) ... 299
PREG_SET_ORDER (Konstante) ... 297
preg_split() (Funktion) ... 199, 299
prev() (Funktion) ... 239
print (Anweisung) ... 100, 184, 941
print_r() (Funktion) ... 188
print() (Anweisung) ... 135
printf() (Funktion) ... 186
Prozedur ... 170
Prozedurale Programmierung ... 305
PSlib ... 877
Pull Request ... 1017, 1020
Punkt-Syntax ... 308
Pyrus ... 1006

Q

qmail ... 509
Quadratwurzel ... 259
query() (Methode) ... 823
quoted_printable_decode() (Funktion) ... 751
quoted_printable_encode() (Funktion) ... 751
quotemeta() (Funktion) ... 215
Quoten ... 215

R

rad2deg() (Funktion) ... 267
rand() (Funktion) ... 264
range() (Funktion) ... 248
re2c ... 66
read() (Methode) ... 730
readFromStream() (Methode) ... 875
Real ... 103
ReCAPTCHA ... 968
Rect() (Methode) ... 863
Referenzen ... 113, 319
reflectionClass (Klasse) ... 357
reflectionExtension (Klasse) ... 357
reflectionMethod (Klasse) ... 357
reflectionObject (Klasse) ... 357
reflectionProperty (Klasse) ... 357
RegEx → Reguläre Ausdrücke
register_globals (php.ini) ... 252, 884
register_long_arrays (php.ini) ... 253, 400
register() (Methode) ... 769
Reguläre Ausdrücke ... 291
Links filtern ... 302
Postleitzahlen ... 300
Telefonnummern ... 302
Rekursive Funktionen ... 175
relaxNG ... 825
relaxNGValidate() (Methode) ... 825
relaxNGValidateSource() (Methode) ... 825
REMOTE_ADDR ... 504
Remoting ... 761
rename() (Funktion) ... 729

require_once() (Anweisung) 98
require() (Anweisung) 96
reset() (Funktion) 239
Resource 103
Resource Limits (php.ini) 933
REST 763, 764, 781
Rethans, Derick 697, 896, 947
return (Anweisung) 98, 170
rewind() (Funktion) 726
rewind() (Methode) 355
RFC (Request for Comments) 1024
rmdir() (Funktion) 729
ROLLBACK (SQL) 544
round() (Funktion) 266
RPC 762
RPM 43
rtrim() (Funktion) 207
Rückgabewerte 170
als Referenz 172
mehrere 171
Runden 266

S

Sajax 795
Salt 912
SAPI 41
save() (Methode) 660, 819, 822, 827, 875
saveHTML() (Methode) 819
saveHTMLFile() (Methode) 819
saveToStream() (Methode) 875
saveXML() (Methode) 819, 822, 827
SAX 808, 809
Schema 805, 825
schemaValidate() (Methode) 825
schemaValidateSource() (Methode) 825
Schleifen 149
do-while 160
Endlos- 151
for 149
für Arrays 234
verschachteln 153
while 154
Schlitt, Tobias 1002
Schlüsselwörter 106
Schreibzugriff 721
Screen Scraping 904
Second-Level-Domain (SLD) 459
SELECT (SQL) 534
sendmail 509
sendmail_from (php.ini) 511, 517
sendmail_path (php.ini) 511
Sequenzen 659
Serialisieren 351
serialize() (Funktion) 483, 732
service() (Methode) 770
Service-Anbieter 762
Service-Konsument 762
Session Hijacking 504, 889
Session Riding 900
session_destroy() 491
session_id() (Funktion) 492
session_name() (Funktion) 491
session_regenerate_id() (Funktion) 505
session_set_save_handler() (Funktion) 496, 499
session_start() (Funktion) 485
session_unset() (Funktion) 491
session.auto_start (php.ini) 485
session.gc_divisor (php.ini) 486
session.gc_maxlifetime (php.ini) 486
session.gc_probability (php.ini) 486
session.name (php.ini) 491
session.referer_check (php.ini) 505
session.save_path (php.ini) 482, 929
session.use_cookies (php.ini) 484
session.use_only_cookies (php.ini) 484
session.use_trans_sid (php.ini) 484
Session-Cookies 461
SessionHandlerInterface (Interface) 503
Sessions 481, 915, 988
<iframe>-Problem 485
Daten auslesen 489
Daten löschen 490
Datenprobleme 492
geschützter Bereich 493
in Datenbanken 495
mit Cookies 483
per URL 483
Performance 485
Session-ID 483
Sicherheit 503
set_include_path() (Funktion) 100
setAttribute() (Methode) 821
setcookie() (Funktion) 460
SetFillColor() (Methode) 863
SetFont() (Methode) 862
setlocale() 285
setlocale() (Funktion) 285
setProperty() (Methode) 826

setrawcookie() (Funktion) 466
settype() (Funktion) 105
SFTP 741
SGML 803
SHA1 911
sha1_file() (Funktion) 226
sha1() (Funktion) 226, 911
Shell-Operator 135
short_open_tag (php.ini) 94, 929
Short-circuit evaluation 129
shuffle() (Funktion) 251
Sicherheit 881
Benutzereingaben 883
JavaScript-Code 888
Lücken 881
Session Hijacking 889
Sessions 503
SQL Injection 890
versteckte Felder 893
XSS (Cross-Site Scripting) 886
Signatur 332
similar_text() (Funktion) 223
Simple API for XML 808
Simple Mail Transfer Protocol 507
Simple Object Access Protocol (veraltet) 763
SimpleXML 765, 808, 814
mit DOM 818
Schreibzugriff 817
XPath 816
simplexml_import_dom() (Funktion) 818
simplexml_load_file() (Funktion) 814
simplexml_load_string() (Funktion) 814
sin() (Funktion) 267
Singleton 387
Sitzung → Sessions
sizeof() (Funktion) 240
SMTP 507
Einstellungen 511
SMTP (php.ini) 511
SMTP Service, Microsoft 510
smtp_port (php.ini) 511
smtp4dev 510
SOA 762
SOAP 763, 767
soapclient() (Funktion) 771
SoapFault (Klasse) 778
SoapServer (Klasse) 777
Sonderzeichen 217
in Cookies 466
sort() (Funktion) 251
soundex() (Funktion) 224
split() (Funktion) 199, 295
spliti() (Funktion) 295
Sprachkonstrukt 100, 182
sprintf() (Funktion) 188
SQL 529
Aggregatfunktionen 541
Aliasse 542
COMMIT 544
COUNT 541
CREATE DATABASE 533
CREATE TABLE 533
Datentypen 531
Datumswerte 532
Joins 540
LIKE 536
ORDER BY 535
ROLLBACK 544
SELECT 534
Stored Procedures 544
Versionen 529
WHERE 536, 537
SQL Developer 651
SQL Injection 890
SQLite 603
Besonderheiten 612
Daten ausgeben 619
Daten bearbeiten 622
Daten löschen 620
Datenzugriff 605
Fehlerbehandlung 607
Installation 604
letzter Autowert 612
PHP-Code einbetten 614
Rückgabewerte 609
SQLite3 604
changes() (Methode) 612
close() (Methode) 606
createFunction() (Methode) 615
escapeString() (Methode) 608, 617
exec() (Methode) 606, 612
lastErrorCode() (Methode) 607
lastErrorMsg() (Methode) 607
lastInsertRowID() (Methode) 613, 617
prepare() (Methode) 608
query() (Methode) 606, 609
querySingle() (Methode) 611, 614
sqlite_exec() (Methode) 606
SQLite3 (Klasse) 606
SQLite3::escapeString() (Methode) 891

SQLite3Result
fetchArray() (Methode) ... 609
SQLite3Result (Klasse) ... 609
SQLite3Stmt
bindValue() (Methode) ... 608
execute() (Methode) ... 608
SQLite3Stmt (Klasse) ... 608
sqlsrv_begin_transaction() (Funktion) ... 634
sqlsrv_commit() (Funktion) ... 634
sqlsrv_connect() (Funktion) ... 629
sqlsrv_execute() (Funktion) ... 633
sqlsrv_fetch_array() (Funktion) ... 631
sqlsrv_fetch_assoc() (Funktion) ... 632
sqlsrv_fetch_object() (Funktion) ... 632
sqlsrv_fetch() (Funktion) ... 632
sqlsrv_get_field() (Funktion) ... 632
sqlsrv_has_rows() (Funktion) ... 636
sqlsrv_num_rows() (Funktion) ... 635, 636
sqlsrv_prepare() (Funktion) ... 633
sqlsrv_query() (Funktion) ... 630
sqlsrv_rows_affected() (Funktion) ... 635
sqlsrv_server_info() (Funktion) ... 637
sqrt() (Funktion) ... 259
srand() (Funktion) ... 251, 264
ssl:// ... 743
Standard Generalized Markup Language ... 803
static ... 178
Statische Variablen ... 178
stderr ... 743
stdin ... 743
stdout ... 743
Stored Procedures ... 544, 635
str_ireplace() (Funktion) ... 204, 213
str_replace() (Funktion) ... 198, 212
str_split() (Funktion) ... 199, 288
str_word_count() (Funktion) ... 195, 201
strcasecmp() (Funktion) ... 222
strcmp() (Funktion) ... 222
stream_context_create() (Funktion) ... 745
stream_filter_append() (Funktion) ... 753
stream_filter_prepend() (Funktion) ... 753
stream_filter_register() (Funktion) ... 751
stream_get_filters() (Funktion) ... 751
stream_get_meta_data() (Funktion) ... 745
Streams ... 742
FTP ... 749
HTTP ... 744
Kompression ... 753
Kontext ... 745
Metadaten ... 745
PHP ... 749
strftime() (Funktion) ... 284
String, Umwandlungstabelle ... 219
string.rot13 (Filter) ... 750
string.strip_tags (Filter) ... 750, 756
string.tolower (Filter) ... 750
string.toupper (Filter) ... 750
Strings ... 103, 191
Ähnlichkeit ... 223
alphabetisch sortieren ... 125
auf Zeichen zugreifen ... 192
beschneiden ... 205
Datum ... 279
entwerten ... 214
formatierte ... 186
Groß- und Kleinschreibung ... 202
heredoc ... 184
konkatenieren ... 120
nowdoc ... 185
Performance ... 199
Position ... 208
suchen & ersetzen ... 207
teilen ... 195
verbinden ... 120
vergleichen ... 222
Verschlüsselung ... 225
Zeilenumbrüche ... 194
strip_tags() (Funktion) ... 220, 750
stripcslashes() (Funktion) ... 215
stripslashes() (Funktion) ... 214, 216
stristr() (Funktion) ... 210
strlen() (Funktion) ... 193, 195
strpbrk() (Funktion) ... 211
strpos() (Funktion) ... 208, 952
strptime() (Funktion) ... 284
strrev() (Funktion) ... 227
strripos() (Funktion) ... 209
strstr() (Funktion) ... 209
strtok() (Funktion) ... 196
strtolower() (Funktion) ... 204, 750
strtotime() (Funktion) ... 280, 596
strtoupper() (Funktion) ... 204, 750
strtr() (Funktion) ... 213
sub() (Methode) ... 289
substr_compare() (Funktion) ... 223
substr_count() (Funktion) ... 210
substr_replace() (Funktion) ... 212
substr() (Funktion) ... 193, 195, 205
Subtraktion ... 117
Superglobale Arrays ... 114, 252
Suraski, Zeev ... 31
SVN ... 1000

switch ... 145
mit Bedingung ... 148
system() (Funktion) ... 729
Systemoperationen ... 729

T

tan() (Funktion) ... 267
TCP ... 743
tcp:// ... 743
TCPDF ... 877
Text_CAPTCHA ... 906
Text_Password ... 983
textContent (Eigenschaft) ... 820
textOut() (Methode) ... 875
time() (Funktion) ... 277
TIMESTAMP (SQL) ... 532
Tortendiagramm ... 844
Traits ... 339
Transaktionen ... 543
transformToDoc() (Methode) ... 826
transformToUri() (Methode) ... 827
transformToXml() (Methode) ... 827
Trennung Code und Inhalt ... 96
Trigger ... 654
trim() (Funktion) ... 207
True Type Fonts ... 842
TTF ... 842
Turing, Alan ... 905, 967
Type Casting → Typkonvertierung
Typkonvertierung ... 105
TYPO3 ... 33

U

Überladen ... 166, 314, 333
Überschreiben ... 721
Überschreiben (OO) ... 317
UBR ... 764
ucfirst() (Funktion) ... 204
ucwords() (Funktion) ... 204, 389
UDDI ... 763, 764, 784
UDP ... 743
udp:// ... 743
Ungleichheit ... 122
genaue ... 122
Uniform Resource Locator → URL
uniqid() (Funktion) ... 226
Unisys ... 838
Universal Description, Discovery and Integration ... 763
unlink() (Funktion) ... 729
unserialize() (Funktion) ... 483
unset() (Funktion) ... 112, 232, 490
URI ... 959
URL ... 33, 221, 483
Informationen ... 959
url_rewriter.tags (php.ini) ... 492
urldecode() (Funktion) ... 221
urlencode() (Funktion) ... 221, 466, 596
US Secure Hash ... 226
use (Schlüsselwort) ... 340, 376
usort() (Funktion) ... 126
UTF-8 ... 218, 812
utf8_decode() (Funktion) ... 690
utf8_encode() (Funktion) ... 688, 693

V

valid() (Methode) ... 355
validate() (Methode) ... 825
var_dump() (Funktion) ... 188, 679, 943
var_export() (Funktion) ... 189, 728
VARCHAR (SQL) ... 532
Variable, Referenzen ... 113
Variablen
als Funktionsnamen ... 173
ausgeben ... 107
Gültigkeit ... 168
Namenskonventionen ... 106
statisch ... 178
variable ... 106
vordefinierte ... 114
Vererbung ... 315
Vergleichsoperatoren ... 121
Verschlüsselung ... 225
DES ... 226
MD5 ... 225
US Secure ... 226
Virtueller Pfad ... 731
Vorausfüllung
Checkboxen ... 433

W

W3C ... 763, 803
Wahrheitswert ... 103, 121
WAMPSERVER ... 47
Warnock, John ... 859

Web Service Description Language 763
Web Services 761, 762
Architektur 762
gzip 766
Installation 768
Pakete 767
Probleme 765
Protokolle 763
Webserver 44
Apache 955
Authentifizierung 915
nicht gefunden 82
Port 45
Serverfehler 88
Websites
Textversion erzeugen 756
WHERE (SQL) 536, 537
while 154
alternative Syntax 156
Whitespace 207, 812
window.alert() (JavaScript) 789
Windows
Authentifizierung 920
Installation 44
PHP kompilieren 53
Winer, Dave 761
--with-apxs[=Datei] (Schalter) 67
--with-config-file-path=/etc/ (Schalter) 70
--with-gd (Schalter) 837
--with-haru[=Bibliothek] (Schalter) 861
--with-mysql (Schalter) 68
--with-mysql=/pfad/zu/mysql (Schalter) 573
--with-mysql=/pfad/zu/mysql_config (Schalter) 574
--with-oci8 (Schalter) 649
--without-pcre-regex (Schalter) 295
--without-pear (Schalter) 72
--with-pdflib[=Bibliothek] (Schalter) 860
--with-regex[=TYPE] (Schalter) 293
--with-xsl[=Pfad] (Schalter) 809
wordwrap (Funktion) 194
WSDL 763, 764
nuSOAP 772
PHP-SOAP 779
WS-Security 766
WWW-Authenticate-Header 916

X

XAMPP 46, 65
Xdebug 947
X-Mailer 514
XML 803
Installation 809
Namensräume 806
Programmierzugriff 808
Schema 805
SimpleXML 814
valide 805
validieren 824, 825
wohlgeformt 803
XML_OPTION_CASE_FOLDING (Konstante) 812
XML_OPTION_SKIP_WHITE (Konstante) 812
XML_OTPION_TARGET_ENCODING (Konstante) 812
xml_parse() (Funktion) 810
xml_parser_create() (Funktion) 809
xml_parser_free() (Funktion) 810
xml_parser_get_option() (Funktion) 812
xml_parser_set_option() (Funktion) 812
xml_set_character_data_handler() (Funktion) 810
xml_set_default_handler() (Funktion) 812
xml_set_element_handler() (Funktion) 810
xml_set_processing_instruction_handler() (Funktion) 812
xml_set_unparsed_entity_decl_handler() (Funktion) 812
XML_SVG 74
XMLHttpRequest 793
XML-RPC 761
xor 129
XPath 807
DOM 823
SimpleXML 816
XQuery 808
XSD 805
XSL 806
XSLT 806, 826
XsltProcessor (Klasse) 826
XSS 886

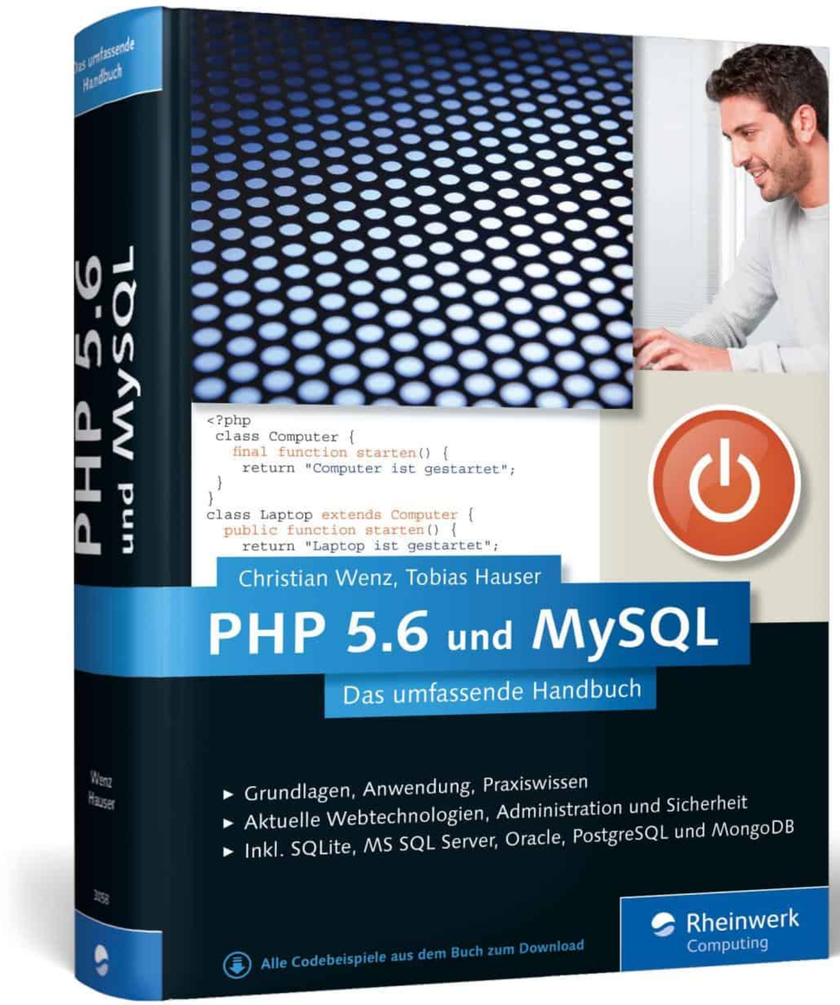
Y

yield 177

Z

Zählervariable 155
Zeichenketten 103
Zeichensätze 218

Zeiger 238
Zeitstempel 277
Zend 32
Debugger 951
Studio 951
Zend Engine 2 305
Zend Framework 380
Factory 389
MVC 382
Singleton 387
ZendPdf 877
Zuweisungsoperator 103, 231, 309





Christian Wenz ist Autor, Trainer und Berater mit Schwerpunkt Webtechnologien und Web-Security. Seine Bücher wurden in rund ein Dutzend Sprachen übersetzt. Er ist gefragter Referent auf Entwicklerkonferenzen im In- und Ausland, Coautor der PHP-5-Zertifizierung und Mitbegründer des PHP Security Consortiums. Als Teilhaber der Agentur Arrabiata Solutions GmbH berät er Firmen auf der ganzen Welt bei ihrer Webstrategie, schult Entwickler und führt Security-Audits durch.



Tobias Hauser schreibt als Buchautor seit vielen Jahren zu allen wichtigen Webthemen. Er veröffentlicht regelmäßig in Fachzeitschriften, spricht auf Entwicklerkonferenzen im In- und Ausland und ist Mitglied der Typo3 Certification Group. Als Teilhaber der Agentur Arrabiata Solutions GmbH realisiert er Web- und Intranetprojekte, führt Usability-Audits durch und realisiert E-Learning-Plattformen.

Christian Wenz, Tobias Hauser

**PHP 5.6 und MySQL –
Das umfassende Handbuch**

1.047 Seiten, gebunden, März 2015
39,90 Euro, ISBN 978-3-8362-3058-2

www.rheinwerk-verlag.de/3714

Wir hoffen sehr, dass Ihnen diese Leseprobe gefallen hat. Sie dürfen sie gerne empfehlen und weitergeben, allerdings nur vollständig mit allen Seiten. Bitte beachten Sie, dass der Funktionsumfang dieser Leseprobe sowie ihre Darstellung von der E-Book-Fassung des vorgestellten Buches abweichen können. Diese Leseprobe ist in all ihren Teilen urheberrechtlich geschützt. Alle Nutzungs- und Verwertungsrechte liegen beim Autor und beim Verlag.

Teilen Sie Ihre Leseerfahrung mit uns!