2. Auflage

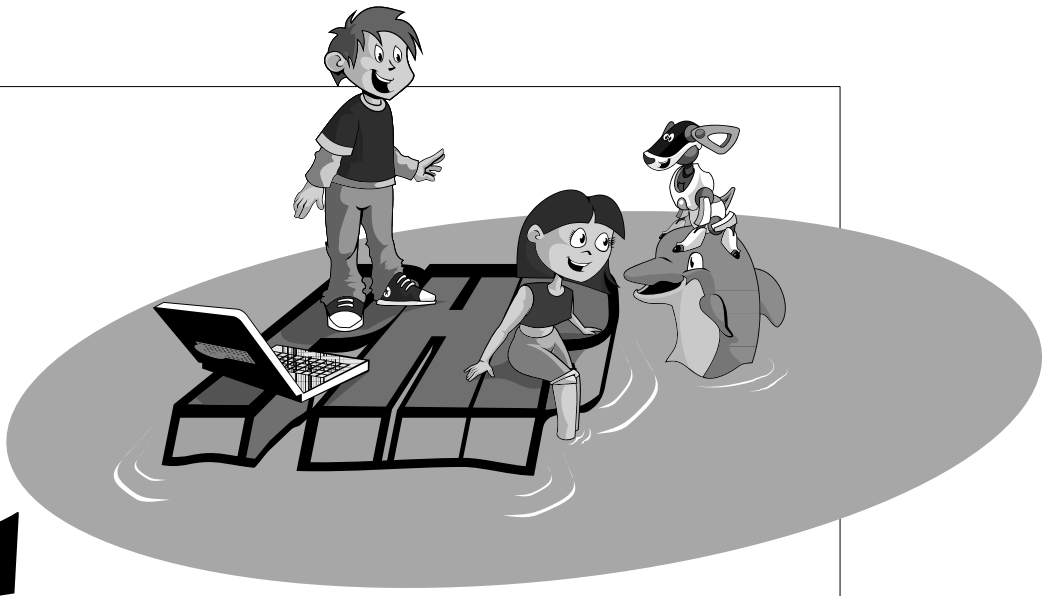
PHP und MySQL Praxisbuch

FÜR KIDS

Johann-Christian Hanke

Auf der CD:
XAMPP mit PHP 5 und MySQL 5, SELFPHP, vier Editoren, sechs CMS (u.a. Joomla) und viele Praxisbeispiele

bhv



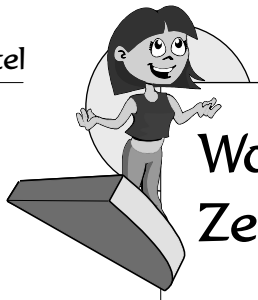
1 Pfiffige Funktionen im Praxiseinsatz

Funktionen, Funktionen, Funktionen ... was wäre eine Programmiersprache ohne diese raffinierten »Sprachgebilde«. Sicher ist dir `mail()` zum Verschicken von Mails längst genauso geläufig wie `date()` zum Anzeigen von Datum und Uhrzeit. Du öffnest und schließt Dateien, setzt Cookies und fragst Datenbanktabellen ab ... natürlich alles per Funktion.

Dieses Kapitel dreht sich um Funktionen für Strings und Arrays. Denn gerade Zeichenketten und Wertelisten spielen in PHP eine wichtige Rolle. Ich stelle dir ausgewählte String- und Array-Funktionen an kleinen Beispielen vor und verrate dir, wie du sie in der Praxis einsetzt. Betrachte das Kapitel als »Denksportrunde zum Aufwärmen«, ehe wir bei den etwas größeren Projekten »unsere Muskeln spielen lassen«.

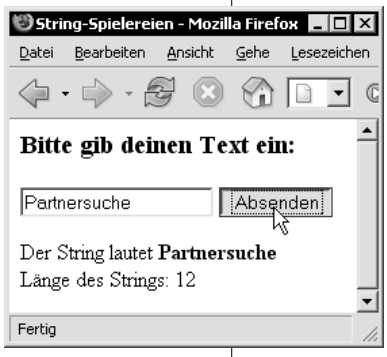
In diesem Kapitel geht es also um folgende Themen:

- Prüfen und Verändern von Zeichenketten
- Unterdrücken überlanger Strings
- Aufteilen von Arrays anhand eines Trennzeichens
- Auslesen von CSV-Textdateien
- Erstellen und Ausgeben von mehrdimensionalen Arrays



Wo steckt der Buchstabe? Zeichenketten prüfen!

Darum prüfe, wer sich ewig bindet ... Nein, nein – dafür ist es vielleicht noch etwas zu früh! In diesem Kapitel prüfen wir nicht unsere Partner, sondern simple Zeichenketten. Denn schließlich hast du es alle Nase lang mit diesen »Dingern« zu tun. Egal ob Suchenfeld, E-Mail-Adresse oder Eintrag in deinem Gästebuch – überall entstehen Zeichenketten mit den Eingaben deiner Besucher. Und da liegt oft so einiges im Argen ...



Im Beispiel führe ich dir ein paar nützliche Zeichenkettenfunktionen zum Testen von Formulareingaben vor. Mache mit! Erstelle eine HTML-Seite mit einem Formular. Ermittle den Inhalt des Texteingabefelds *eingabe* und werte diesen String aus.

Länge einer Zeichenkette ermitteln

Schreibe dafür das nachfolgende Beispiel ab – ich zeige dir den Code zwischen den Tags `<body>` und `</body>`. Dabei bereitest du den entsprechenden »Experimentierbereich« für die Funktionen schon vor:

```
<h3>Bitte gib deinen Text ein:</h3>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>"
      method="post">
<input type="text" name="eingabe">
<input type="submit" value="Absenden">
</form>
<p>
<?php
if (isset($_POST['eingabe'])) {
  $eingabe = $_POST['eingabe'];
  echo "Der String lautet <strong>$eingabe</strong>";
  echo "<br>Länge des Strings: " . strlen($eingabe);
}
?>
</p>
```

Hast du es gemerkt? Ich verwende die schon im ersten Band vorgestellte Funktion `strlen()`. Sie ermittelt die Länge einer Zeichenkette – ideal zum Festlegen der Mindestlänge. Als Argument übergibst du den gewünschten String – im Beispiel den Wert der Variablen `$eingabe`.

Die Grundsyntax der Funktion sieht übrigens folgendermaßen aus:

```
int strlen(string str)
```

Warum reite ich auf dieser Syntax so herum? Weil es genau die Schreibweise ist, die du auch im offiziellen PHP-Handbuch findest. Schlage nach! Du findest diese Hilfedatei schließlich auf der CD zum Buch bzw. als Download unter *www.php.net/download-docs.php*.

Das `int` steht dabei für Integer, also einen Zahlwert. Die Funktion gibt einen Zahlwert zurück, z.B. 12. Die Zeichenfolge `str` steht für das Argument, im Beispiel die Variable `$eingabe`. Und mit `string` ist der Typ des Arguments gemeint – es muss ein String sein.

Uuups, mein kleines »Wiederholungsbeispiel« hat dich kalt erwischt? Alles ging etwas zu schnell und war zu kryptisch? Bei dir schleichen sich schon an dieser Stelle aberwitzige Fehlermeldungen ein, die dir den Spaß an den Funktionen verderben? Schaue auf die CD zum Ordner `beispiele`! Hier liegen alle Beispieldateien wohlgeordnet nach Kapiteln und du kannst dir das leidige Abschreiben sparen.

Überschüssige Leerzeichen entfernen

Mache dir doch einmal den Spaß und tippe einen einzigen Buchstaben in das Formularfeld – z.B. ein *P* wie *PHP*. Setze davor allerdings drei Leerzeichen. Prompt liefert dir `strlen()` eine *4* für vier Zeichen. Auch Leerzeichen werden also mitgezählt, selbst die am Anfang und am Ende.

Das ist nicht optimal, denn schon ein einziges Leerzeichen zuviel kann bei einer Auswertung stören! So wundert sich dein Besucher vielleicht, wenn die Suchfunktion nicht fündig wird, weil sie die versehentlich gesetzten Leerzeichen mitsucht. Weg mit den Dingern! Das besorgt die Funktion `trim()` mit folgender Syntax:

```
string trim(string str)
```

Diese Funktion entfernt alle Leerzeichen, Zeilenumbrüche oder Tabsprünge am Anfang und Ende einer Zeichenkette. Probiere es aus: Erweitere unser kleines Auswerteskript um folgende Zeile und notiere sie als zweite Zeile innerhalb der geschweiften Klammern:

```
$eingabe = trim($eingabe);
```

Du möchtest nur die Leerzeichen bzw. Umbrüche *am Anfang* entfernen? Verwende `ltrim()` mit folgender Syntax: `string ltrim(string str)`. Die Funktion `rtrim()` dagegen wirkt nur auf das Ende der Zeichenkette.

Position eines Zeichens ermitteln

Nachdem wir unsere Strings also »getrimmt« und »vermessen« haben, können wir zum Spaß die Position eines Zeichens in einer Zeichenkette ermitteln. So findest du z.B. heraus, ob die Eingabe einen Klammeraffen enthält und an welcher Stelle dieser (das erste Mal) vorkommt. Das gelingt mit `strpos()` – die Syntax dieser Funktion sieht allerdings verboten aus:

```
int strpos(string haystack, string needle [,int offset])
```

Die eckigen Klammern sind schon einmal gut: Sie weisen darauf hin, dass dieses Argument *optional* ist. Mit anderen Worten: Du kannst es weglassen. Es handelt sich dabei übrigens um den Startwert, ab dem gesucht werden soll, den *offset*. Das `int` davor bedeutet, dass dieser Wert eine Ganzzahl sein muss. Soll uns aber egal sein – wir ignorieren das Argument.

Interessant sind dagegen die Strings *haystack* und *needle*. Hier wird tatsächlich die »Nadel im Heuhaufen« gesucht. Na dann suchen wir doch mal nach dem Klammeraffen – ergänze einfach folgende Zeile am Schluss des PHP-Abschnitts:

```
echo "<br>Position des @: " . strpos($eingabe,"@");
```

Und nun gibst du zur Probe eine Mail-Adresse ein wie `info@phpkid.de`. Die Funktion ermittelt eine 4, obwohl das Zeichen an fünfter Stelle steht?

Vergiss nicht, dass jegliche »Zählerei« bei Funktionen und Arrays in der Regel bei 0 beginnt! Das ist auch bei `strpos()` nicht anders! Hättest du das @-Zeichen am Anfang notiert, hätte die Funktion eine 0 ausgespuckt. Denn die 0 steht für die erste Position in einem String.

Nichtvorhandensein des Zeichens prüfen

Du bist weniger daran interessiert, an welcher Stelle das Zeichen vorkommt? Du willst eher herausfinden, *ob* es überhaupt vorhanden ist? Gut zu wissen, dass `strpos()` – wie viele andere Funktionen auch – bei Misserfolg den booleschen Wert (Wahrheitswert) *false* zurückgibt.

Doch was erlebst du, wenn du folgenden Test verwendest?

```
if (!strpos($eingabe,"@")) {
  echo "<br>Kein Klammeraffe enthalten!";
}
```

Richtig – einen Reinfall! Schon, schon ... die Notation ist korrekt. Der Negationsoperator ! sorgt dafür, dass das Ergebnis der Funktion nicht wahr sein darf. Schließlich handelt es sich dabei um die vollkommen legitime Abkürzung von `strpos($eingabe,"@") == false`. Aber auch diese Notation rettet dich nicht vor dem Desaster. Der Haken: Auch eine 0 wird als *false* angesehen. Genau wie eine 1 schließlich als *true* gewertet wird. Null und Eins, true und false – das ist (nicht nur in PHP) ein und dasselbe. Und wenn der Klammeraffe an erster Stelle auftaucht, gibt `strpos()` nun einmal die 0 zurück. *Kein Klammeraffe enthalten* lautet die Auskunft unserer if-Abfrage in diesem Fall – eine pure Falschmeldung.

Nutze statt des doppelten Gleichheitszeichens einfach das dreifache! Mit === prüfst du auf »absolute Übereinstimmung«, auf *Identität*! Dabei wird auch der Typ der zu vergleichenden Werte berücksichtig. Während == zulässt, dass der Integerwert 0 dem Wahrheitswert *false* gleichgesetzt wird, ist das bei === eben nicht mehr zulässig. Der Typ *Integer* stimmt nicht mit dem Typ *Boolean* (Wahrheitswert *true* bzw. *false*) überein!

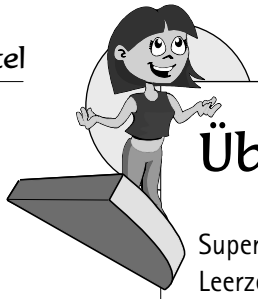
Notiere Folgendes, um zuverlässig auf Nichtvorhandensein zu reagieren:

```
if (strpos($eingabe, "@") === false) {
  echo "<br>Kein Klammeraffe enthalten!";
}
```

Kleines Suchskript programmieren

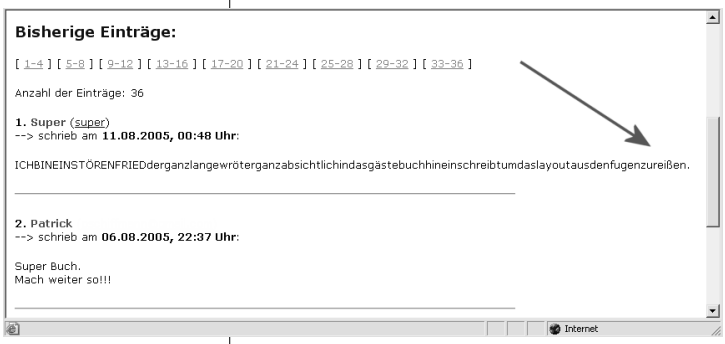
Übrigens kannst du mit `strpos()` nicht nur nach einzelnen Zeichen suchen. Auch längere Zeichenketten und selbst ganze Wörter lassen sich im Argument `needle` eintragen. Dadurch eignet sich diese Funktion durchaus zum Erstellen eines kleinen Suchskripts wie hier in der `minisuche.php`:

```
<?php
$nadel = "Heu"; // Hier Suchwort eintragen
$heuhaufen = "Das ist ein großer Haufen Heu";
if (strpos($heuhaufen, $nadel) > 0) {
  echo "Wort <b>$nadel</b> enthalten in:";
  echo "<div>$heuhaufen</div>";
} else {
  echo "Wort <b>$nadel</b> nicht enthalten!";
  }
?>
```



Überlange Zeilen auftrennen

Supernervig: Diese Zeitgenossen, die absichtlich ellenlange Wörter ohne Leerzeichen in deine Gästebücher schreiben. Mit Absicht – sie wollen dein Layout zerreißen. Wie hässlich das aussehen kann, zeigt die Abbildung.



Nervt total: Gästebuchschreiber, die nur stören wollen

Aber es gibt eine Lösung für dieses Problem!

Die Funktion wordwrap()

Zum einen benötigen wir die Funktion `wordwrap`. Per Voreinstellung setzt diese nach dem 75sten Zeichen einen Umbruch per new line (`\n`):

```
$neuertext = wordwrap($altertext);
```

Du musst lediglich die Anzahl der Zeichen und den Typ des Umbruchs steuern. Das gelingt mit den Parametern *width* und *break*. Hier also die volle Syntax in ganzer Pracht:

```
string wordwrap(string str [,int width [,string break [,bool cut]]])
```

Die Zeile `$eingabe = wordwrap($eingabe, 60, "<br>");` setzt z.B. nach 60 Zeichen einen HTML-Umbruch, den Break (`<br>`). Gibt es bei der Geschichte auch einen Haken? Leider!

Überlange Wörter werden dabei nicht getrennt. Das gelingt erst, wenn du als letzten, optionalen Parameter *cut* eine 1 notierst. Aber auch das ist leider nicht die Lösung. Denn dann werden *alle* Strings nach der jeweiligen Zeichenzahl getrennt. Egal, ob es mitten im Wort ist oder nicht.

Mit anderen Worten: Die Funktion `wordwrap()` alleine löst unsere Probleme leider nicht. Wir müssten jedes einzelne Wort nehmen, prüfen und ggf. bearbeiten. Wie machen wir das? Wir kramen `explode()` und `foreach()` aus der »Erinnerungsschublade« und fragen damit jedes einzelne Wort ab.

explode() und foreach()

Die Syntax von `explode()` und `foreach()` ist dir vertraut? Sonst schlage schnell im Vorgänger oder im schon erwähnten PHP-Handbuch nach. Die Funktion `explode()` wandelt eine Zeichenfolge mithilfe eines Trennzeichens in ein Array um. Mit `foreach()` durchläufst du dieses Array.

Und hier nun der Quellcode für meinen »Langstringkiller«, Version 1. Ich zeige dir den Bereich zwischen den Tags `<body></body>`. Du findest den Code im Ordner `kapitel01` unter dem Namen `langkiller1.php`:

```
<h1>Lange Strings teilen</h1>
<p>
<?php
$string1 = "Gleich <b>kommt</b> ein
      ganzlangerStringohnePunktundKommaderdasganzeLayout
      zerreißt und das ist nicht schön";
$ausgabe = explode(" ", $string1);
$string2 = "";
foreach ($ausgabe as $value) {
  $string2 .= wordwrap($value, 40, " ", 1) . " ";
}
echo trim($string2);
?>
</p>
```

Die Variable `$string1` beherbergt neben ein paar »normalen« Worten eins dieser berüchtigten »langen Stringmonster«. Getrennt werden die Worte – wie in einem Satz so üblich – durch ein Leerzeichen. Danach greift die Funktion `explode()` und verwandelt diese Zeichenfolge in ein Array. Als Trennzeichen dient das Leerzeichen – macht ja auch Sinn. Auf diese Weise landen alle Wörter im Array `$ausgabe`.

Hat geklappt: Zwangsleerzeichen nach 40 Zeichen!

In der nächsten Zeile initialisiere ich die Variable `$string2` mit einem Leerstring und beginne dann mit der Schleife. Diese durchläuft alle Elemente des Arrays und hält nacheinander alle Werte in der Variablen `$string2` fest. Dabei werden die Wörter natürlich keinesfalls einfach so aneinandergehängt. Denn an dieser Stelle sorgt endlich `wordwrap()` dafür,

dass überlange String durch ein Zwangsleerzeichen unterbrochen werden. Und zwar nach 40 Zeichen! Beachte, dass die bei der Array-Konvertierung »verlorenen« Leerzeichen am Ende auch wieder ergänzt werden müssen, und zwar durch `. " "`! Nach Beendigung der Schleife wird der fertige String dann ausgegeben. Nicht ohne dabei mit `trim()` die überschüssigen Leerzeichen am Anfang und vor allem am Ende zu entfernen.

Eigene Funktion schreiben

Im nächsten Beispiel habe ich alles in eine selbst gestrickte Funktion gepackt. Das ist praktisch. Die Funktion kannst du einfach in eine Funktionsbibliothek stecken – z.B. in die im vorigen Band verwendete Datei `function.inc.php`. So steht sie für alle Projekte bereit. Doch hier ist erst einmal die Eigenbau-Funktion – sie heißt im Beispiel `longkicker()`:

```
<?php
$string = "Gleich <b>kommt</b> ein
      ganzlangerStringohnePunktundKommaderdasganzeLayout
      zerreißt und das ist nicht schön";
function longkicker($arg)
{
  $ausgabe = explode(" ", $arg);
  $arg = "";
  foreach ($ausgabe as $value) {
    $arg .= wordwrap($value, 40, " ", 1) . " ";
  }
  return trim($arg);
}
echo longkicker($string);
?>
```

Wie die Funktion arbeitet, ist dir klar? Als Argument übernimmt sie den zu behandelnden String – im Beispiel `$arg`. Dieser läuft durch die »foreach-wordwrap-Mühle«, wird »getrimmt« und per `return` zurückgegeben.

Wenn du die Zeichenfolge letztlich ausgeben willst, verwendest du einfach die selbstgebastelte Funktion: `echo longkicker($string);` Ideal zum nachträglichen Einbau in Gästebücher, Foren etc.!

Du wunderst dich über meine Klammersetzung bei Funktionen? Tipps zur optimalen Syntax gebe ich dir auf Seite 48–49.

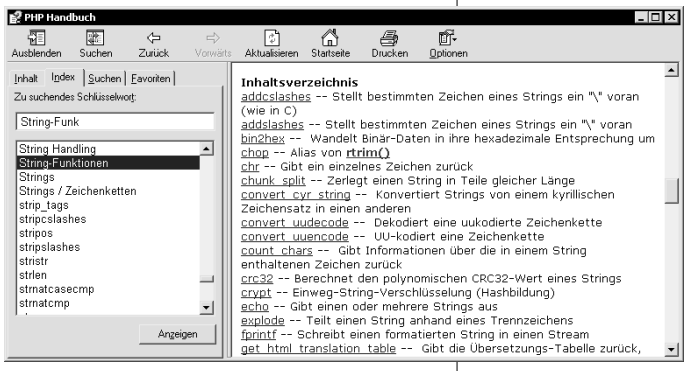
Weitere coole String-Funktionen

Ein Leser des ersten Bandes war zwar begeistert – er hätte sich aber mehr Funktionen gewünscht. Dabei hatte ich mich ganz absichtlich beschränkt! Schließlich wollte ich keine Funktionsreferenz schreiben, sondern durch Beispiele überzeugen. Überlassen wir das Auflisten *aller* Funktionen einfach den »kiloschweren« Schwarten.

Nachschlagen im PHP-Manual

Ich erinnere immer wieder daran – solche eine »Schwarte« haben wir auch auf der CD. Und die ist vom Inhalt her zwar »schwergewichtig« – wiegt ansonsten aber nur 16 Gramm. Und so schlägst du im »Handbuch« nach:

- Öffne die Hilfedatei *php_manual_de.chm* – in dieser Datei findest du das PHP-Handbuch.
- Gehe ins Register *Index* und tippe das Suchwort *String-Funktion*.
- Doppelklicke links auf den gleichnamigen Eintrag und rolle im rechten Hilfefenster ein Stück nach unten, bis der Text *Inhaltsverzeichnis* erscheint.



Ausführlich und mit vielen Beispielen: Das PHP-Handbuch auf der Buch-CD

Und schon hast du eine wunderbare Funktionsreferenz für alle Zeichenkettenfunktionen. Links steht die Funktion und rechts daneben findest du eine kleine Erläuterung. Ich nutze diese Übersicht sehr häufig, wenn ich eine ganz bestimmte Funktionalität benötige.

Natürlich möchte ich dir auch (noch einmal) die SELFPHP von Damir Enseleit ans Herz legen. Auch hier findest du eine ausführliche Funktionsreferenz und ein großes Befehlsverzeichnis. Surfe zu *www.selfphp.de* oder schaue auf die CD zu diesem Buch! Dort liegt sie in einer Hilfedatei.

Trotzdem geht es meist nicht ohne Probieren ab. Denn oft gibt es mehrere Funktionen, die den gleichen Zweck erfüllen. Und manchmal braucht man da schon den richtigen Tipp, um die passende Funktion zu finden. Deshalb stelle ich dir auch in diesem Buch immer wieder neue Funktionen vor. Fangen wir gleich damit an!

Ausgewählte String-Funktionen

Als Nächstes folgen ein paar ausgewählte String-Funktionen bzw. Anweisungen, von denen dir zumindest einige schon bekannt sind:

Funktion/Anweisung (mit Beispiel)	Erläuterung

`echo`
`echo "Wie geht es $Name?";`
`echo "Hallo {$_POST['Name']}";`
`echo 'Guten Tag, Klaus!';`
	Gibt einen String aus – die runden Klammern werden weggelassen, da `echo` ein Sprachkonstrukt und keine echte Funktion ist.

`nl2br()` `$text = nl2br($text);`	Wandelt `\n`-Zeilenumbrüche im Quelltext zusätzlich um in HTML-Zeilenumbrüche ` `.
`htmlspecialchars()` `echo htmlspecialchars($text);`	Ersetzt die Zeichen `<`, `>`, `&` und `"` durch die entsprechenden HTML-Entities `&lt;`, `&gt;`, `&amp;` und `&quot;`. Das Gegenstück ist `htmlspecialchars_decode()`
`stripslashes()` `$body = stripslashes($body);`	Entfernt die Escape-Zeichen `\`, die von PHP z. B. zum Maskieren von einfachen Gänsefüßchen verwendet werden.
`addslashes()` `$body = addslashes($body);`	Fügt vor `\`, `"`, `'` und `Null` (Null-Byte) einen extra Backslash zum Escapen ein. **Normalerweise nicht nötig**! PHP erledigt das automatisch dank der Einstellung `magic_quotes_gpc = On` in der Datei `php.ini`! (`gpc` wie GET, POST, COOKIE)

`str_replace()`
`$text = "Der Weg führt zu dir!";`
`echo str_replace("Weg","Pfad",`
`$text);`
`// ersetzt Weg durch Pfad`
	Ersetzt in einem String (hier `$text`) eine Passage durch eine andere. Berücksichtigt Groß- und Kleinschreibung (GKS). Hinweis: `str_ireplace()` macht das Gleiche ohne Berücksichtigung der GKS.

`strstr()`
`$email = "test@lexi.de";`
`$ausgabe = strstr($email, "@");`
`echo $ausgabe;`
`// gibt @lexi.de aus`
	Sucht Nadel (hier `@`) im Heuhaufen (hier `$email`) und gibt Reststring ab erstem Fund von Nadel zurück. Bei Nichterfolg gibt die Funktion `false` zurück. Das Gegenstück `stristr()` ignoriert GKS.

`strtolower()`
`echo strtolower("HanSwursT");`
`// gibt hanswurst aus`
	Wandelt alle Großbuchstaben eines Strings in Kleinbuchstaben um, das Gegenstück dazu heißt `strtoupper()`.

Funktion/Anweisung (mit Beispiel)	Erläuterung

`str_repeat()`
`echo str_repeat("so", 5);`
`// gibt` *`sososososo`* `aus`
	Wiederholt die Ausgabe eines Strings (hier `so`) unter Verwendung des festgelegten Faktors (hier `5`).

`substr_count()`
`$text = "Du bist Christian!";`
`echo substr_count($text,"ist");`
`// gibt 2 aus`
	Zählt das Vorkommen einer Zeichenkette (hier `ist`) in einer anderen Zeichenkette (hier Inhalt von `$text`) und gibt die Zahl der Fundstellen aus.

`str_shuffle()`
`echo str_shuffle("hallo");`
`// gibt z.B.` *`laloh`* `aus`
	Mischt eine Zeichenkette und setzt sie anders wieder zusammen. Ähnliches Prinzip wie beim MP3-Player. ;-)

Die zweite Dimension bei Arrays

Die zweite Dimension – so heißt die erste Folge von »Adolars phantastischen Abenteuern«. Adolar landet auf dem Scheibenplaneten – einem »platten« Planeten ohne dritte Dimension. Arrays sind noch platter, sie haben normalerweise nicht einmal die zweite. Doch das ändern wir jetzt!

Du kannst in PHP problemlos innerhalb eines Arrays ein weiteres Array anlegen: So baust du ein Array mit einer weiteren Dimension. Selbst das Mischen von indizierten und assoziativen Arrays ist dabei möglich!

Dein Team – als Datenfeld

Du brauchst solche mehrdimensionalen Arrays zwar selten, trotzdem solltest du die Technik kennen! Im Beispiel speichern wir mehrere Teammitglieder in so einem Multi-Datenfeld ab. Doch bevor wir uns in höhere Dimensionen aufschwingen, zeige ich dir erst einmal ein Wiederholungsbeispiel in der »ersten Dimension«. Vergleiche mit der Datei `array1.php`:

```
$member = array();
$member[0] = "Hannes";
$member[1] = "Tamara";
$member[2] = "Alfredo";
$member[3] = "Clarissa";
echo "Zahl der Einträge: " . count($member);
echo "<br>Viertes Mitglied (Index 3): " . $member[3];
```

Du siehst ein simples, indiziertes Datenfeld mit vier Teammitgliedern. Mehr als einen Wert speichern kannst du nicht – hier den Namen.

Du wunderst dich über die erste Zeile? Diese erzeugt das Array mit Hilfe des Sprachkonstrukts `array()`. Das ist übervorbildlich und keine Pflicht – du kannst diese Zeile also auch weglassen. Sobald du die eckigen Klammern notierst, weiß PHP ganz von allein, dass du ein Array erzeugen willst!

Unterhalb der Feldvariablen ermittle ich die Anzahl der Elemente und gebe das vierte Mitglied aus.

Nicht vergessen: Bei indizierten Arrays beginnt die Zählung bei 0!

So weit – so unspektakulär. Doch wenn du neben dem Namen auch Alter und Hobby der Mitglieder speichern möchtest? Dann kommt dir die zweite Dimension gerade recht!

Zweidimensionales Array erzeugen

Schau dir mal ganz in Ruhe die nächste Variante (`array2.php`) an. Hier hängen wir an das erste eckige Klammernpaar einfach ein zweites dran!

```
$member[0]['name'] = "Hannes";
$member[0]['alter'] = 12;
$member[0]['hobby'] = "Skaten";
$member[1]['name'] = "Tamara";
$member[1]['alter'] = "14";
$member[1]['hobby'] = "Musizieren";
$member[2]['name'] = "Alfredo";
$member[2]['alter'] = 15;
$member[2]['hobby'] = "Tanzen";
$member[3]['name'] = "Clarissa";
$member[3]['alter'] = 11;
$member[3]['hobby'] = "Schwimmen";
```

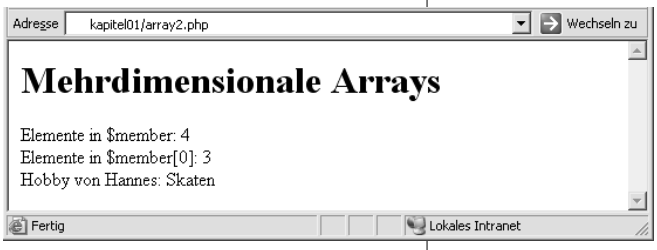
Das erste Array steckt in `$member`. Hier werden die einzelnen Mitglieder gespeichert. Es handelt sich um das altbekannte, indizierte Array. Doch jedes Mitglied wird zu einem weiteren Array. So ist `$member[0]` genau wie `$member[1]` oder `$member[2]` ein eigenständiges, assoziatives Datenfeld. Die Schlüssel heißen `name`, `alter` und `hobby`. Und schon bist du in der zweiten Dimension!

Array-Elemente ausgeben

Mein Musterquelltext zeigt dir natürlich auch, wie du nun auf die einzelnen Elemente der Arrays zugreifst:

```
echo 'Elemente in $member: ' . count($member);
echo '<br>Elemente in $member[0]: ' . count($member[0]);
echo "<br>Hobby von " . $member[0]['name'] . ": " .
     $member[0]['hobby'];
```

Um das Hobby des ersten Mitglieds zu ermitteln, schreibst du im Beispiel `$member[0]['hobby']`. Besonders interessant ist auch der Zähltest per `count()`, den wir zu Anfang durchführen. Wir zählen diesmal einfach die Elemente von zwei Arrays. Auf die Verwendung des Sprachkonstrukts `array()` habe ich übrigens großzügig verzichtet.



Array-Elemente durchzählen – für »$member« sind es 4 und für »$member[0]« 3 Einträge

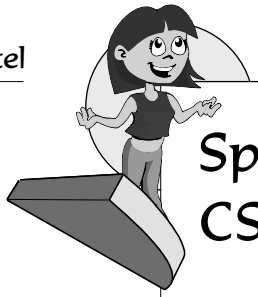
Dir ist der Nutzen dieser Technik noch nicht klargeworden? Ab Seite 82 zeige ich dir im Zusammenhang mit Sessions, wie du die Bewegungsdaten deiner Benutzer mit einem zweidimensionalen Array speicherst.

Ohne Index – das geht auch!

Zum Schluss dieses Kapitels noch ein kleiner Trick zu indizierten Arrays. Du möchtest, dass die *Schlüssel* (also `0`, `1`, `2` usw.) *automatisch* erzeugt werden? Setze nur das eckige Klammernpaar und trage nichts ein. So kommst du auch mit dem Zählen nicht durcheinander:

```
$member[] = "Hannes";
$member[] = "Tamara";
$member[] = "Alfredo";
$member[] = "Clarissa";
```

Allerdings funktioniert der Trick nicht mit unserem zweidimensionalen Array von der Vorseite. Voraussetzung für diese Syntax ist eine fortlaufende, aufsteigende Zählung, bei der sich die Schlüssel nicht wiederholen.



Spaß mit Komma: Auslesen einer CSV-Datei

Schon mal was von CSV gehört? Nein, nicht vom Cuxhavener SV Rot-Weiß, ich meine Comma Separeated Values – also kommagetrennte Werte. Wobei das Komma auch ein Semikolon sein darf.

Hinter CSV-Dateien verbergen sich Tabellen, die in Form von Textdateien gespeichert werden. Jede »Tabellenzeile« wird zu einer Zeile. Eine Spalte wird von der nächsten durch ein Trennzeichen separiert (separated). Dieses Trennzeichen kann ein Komma (comma), ein Semikolon (colon) oder auch ein beliebiges anderes Zeichen sein. Der Inhalt der einzelnen Spalten steht oft in Gänsefüßchen – zumindest wenn es Strings sind.

In diesem Kapitel zeige ich dir, wie du derartige Textdateien ausliest und als HTML-Abschnitt darstellst. Ideal als Ersatz für eine Datenbanktabelle. Ideal selbst für ein klitzekleines Content-Management-System. Soviel vorweg: Dabei haben wieder die Arrays ihre Hände im Spiel.

CSV-Datei inhalt.csv

Was wollen wir überhaupt machen? Wir erstellen eine Datei mit drei Spalten: Überschrift, Einleitung und Hauptbereich. Die Überschrift soll später innerhalb von `<h1></h1>`, die Einleitung innerhalb von `<p><strong> </strong></p>` und der Hauptbereich innerhalb von `<div></div>` ausgegeben werden. Schau dir einfach die nebenstehende Beispielabbildung an!

Als Grundlage dafür dient eine CSV-Datei. Ich habe sie im Beispiel unter dem Namen `inhalt.csv` abgelegt und mit drei Zeilen gefüllt:

```
"Überschrift a","Kurze Einleitungspassage","Textabsatz a"
"Überschrift b","Kurze Einleitungspassage","Textabsatz b"
"Überschrift c","Kurze, knappe Passage ","Textabsatz c"
```

In der Praxis dürfen die einzelnen Zeilen natürlich auch länger sein. Selbst der Inhalt einer langen Passage passt dort hinein!

Datei auslesen per fgetcsv()

Und hier nun das erfreulich kurze Skript zum Auslesen dieser Datei – ich zeige dir lediglich den PHP-Bereich. Die Datei findest du auch auf der CD unter dem Namen `csv_ausgabe.php`:

```
<?php
$fp = fopen("inhalt.csv", "r");
while ($line = fgetcsv($fp, 10000, ",", "\"")) {
  echo "<h1>$line[0]</h1>
<p><strong>$line[1]</strong></p>
<div>$line[2]</div>";
}
  fclose($fp); // Datei wieder schließen
?>
```

Zuerst lese ich den Inhalt der CSV-Datei mit `fopen()` ein und speichere das Ganze in der Variablen `$fp`. Diese dient als Dateizeiger. Dass ich dabei den Read-Only-Modus (Argument `"r"`) verwende, ist sicher klar. Schließlich soll die Datei nur gelesen, aber nicht geschrieben werden. Als Nächstes hat schon die neue Funktion `fgetcsv()` ihren großen Auftritt.

Die Funktion `fgetcsv()` liest eine Datei von der Position des Dateizeigers *zeilenweise* aus und prüft sie auf Komma- bzw. Trennzeichen-getrennte Werte. Sie gibt ein Array zurück. Hier die Grundsyntax:

```
array fgetcsv(resource handle, int length
      [, string delimiter [, string enclosure]])
```

Hinter dem ersten Argument verbirgt sich der Dateizeiger, im Beispiel `$fp`. Das zweite Argument namens `length` ist ein Integer-Wert, eine Zahl. Hier notierst du die maximale Zahl an Bytes, die pro Zeile berücksichtigt werden soll. Ich empfehle `10000`, das sollte genügen. (Längere Strings werden dann beim Auslesen jedoch abgeschnitten.) Der `delimiter` ist wieder ein String – es handelt sich um das Trennzeichen. Wenn du dieses Argument weglässt, nimmt die Funktion automatisch das Komma. Wir lassen das Argument jedoch nicht weg: Es ist wichtig, dass du das Trennzeichen sehen und ggf. verändern kannst. Auch das letzte Argument ist optional – es wurde in PHP 4.3 eingeführt. Hinter `enclosure` verbirgt sich das Zeichen, mit welchem die Strings eingeschlossen sind. Schau dir meine CSV-Datei an: Ich habe das Gänsefüßchen gewählt. Nimm zur Probe das Gänsefüßchenpaar weg. Auch das ist nicht tragisch, solange du kein Komma in deiner Passage verwendest.

Die `while`-Schleife dient dazu, die gesamte Datei auszulesen. Denn `fgetcsv()` wirkt – wie schon erwähnt – immer nur zeilenweise. Was passiert im ersten Durchlauf? Wozu die Variable `$line`? Ganz einfach – um die Werte der ersten Zeile zu sichern. Beim ersten Durchlauf speichert das Array `$line` die Werte aus der ersten Zeile – das sind `$line[0]`, `$line[1]` und `$line[2]`. Diese gibt unser Skript in hübsch formatierter Weise aus. Danach beginnt die Schleife erneut und macht sich über die nächste Zeile her. Erst wenn die Datei fertig ausgelesen ist, wird `while()` gestoppt.

Zusammenfassung

Geschafft! Du hast dich von Funktionen und Arrays nicht aus der Ruhe bringen lassen.

- Du kennst die Funktionen `strlen()` zum Ermitteln der Länge von Zeichenketten oder `trim()` zum Entfernen überschüssiger Leerzeichen am Anfang und Ende eines Strings.
- Du verwendest `strpos()` zur Ermittlung der Position eines Zeichens oder Strings (Needle) in einem anderen String (Haystack). Du hast dich mit der im PHP-Handbuch verwendeten Grundsyntax von Funktionen beschäftigt, für `strpos()` lautet sie beispielsweise so:
  `int strpos(string haystack, string needle [,int offset])`
- Du kennst den Vergleichsoperator `===` zum Prüfen auf Identität, auf absolute Übereinstimmung. Auf diese Weise kann eine `0` nicht mehr als *false* interpretiert werden.
- Du trennst überlange Strings mit den Funktionen `wordwrap()` und `explode()` in Zusammenarbeit mit einer `foreach`-Schleife.
- Du hast das Schreiben eigener Funktionen wiederholt, und zwar am Beispiel eines »Stringtrenners« zum Trennen überlanger Strings.
- Du weißt, dass du alle Stringfunktionen im PHP-Handbuch nachschlagen kannst. Dieses liegt als bequeme HTML-Hilfedatei auf der Buch-CD.
- Du weißt, dass du in PHP mit mehrdimensionalen Arrays arbeiten kannst. Füge zum ersten Schlüssel einfach einen zweiten hinzu und speichere so ein Array in einem Array: `$member[2]['alter']`
- Du kennst die Funktion `fgetcsv()` zum Auslesen von Trennzeichen-separierten Textdateien. Du übergibst ihr Dateizeiger, maximale Zeilenlänge, Trennzeichen und ggf. die Zeichen, mit denen die Abschnitte umschlossen sind: `fgetcsv($fp, 10000, ",", "\"")`

Ein paar Fragen ...

Frage 1: Wie nennt man den Wert, der bei Arrays in eckigen Klammern notiert wird? Tipp: Es gibt mindestens drei Bezeichnungen dafür.

Frage 2: Welche Funktion zählt das Vorkommen einer Zeichenkette in einer anderen Zeichenkette?

Frage 3: Welche Funktion wandelt alle Großbuchstaben einer Zeichenfolge komplett in Kleinbuchstaben um?

Frage 4: Du möchtest zwei Werte vergleichen und dabei auf Identität überprüfen. Welchen Operator setzt du ein?

Frage 5: Du möchtest die Anzahl der Elemente in `$_POST` ausgeben. Wie sieht der entsprechende Code aus?

Frage 6: Wofür steht die Abkürzung CSV?

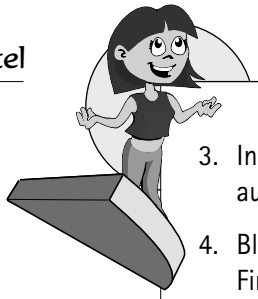
Hinweis: Die Antworten zu den Fragen findest du grundsätzlich auf der CD im Ordner `fragen`.

... und ein paar Aufgaben

1. Schreibe ein Formular mit einem einzigen Feld für die E-Mail-Adresse. Dieses soll nach Klick auf *Absenden* den Inhalt an sich selbst schicken. Das dort vorhandene Skript soll: a) versehentliche Leerzeichen am Anfang und Ende der Eingabe entfernen, b) prüfen, ob der String ein @ und einen Punkt **.** enthält und c) prüfen, ob er länger als 7 Zeichen ist. Wenn alle diese Bedingungen zutreffen, soll das Skript ausgeben: *Danke für die Eingabe*. Ansonsten soll jedoch *Eingabe nicht korrekt!* eingeblendet werden. Nenne die Datei `emailcheck.php`.

2. Erstelle eine PHP-Seite namens `stadt.php` und notiere dort untenstehendes Array. Schreibe eine Schleife, mit der du *Key* und *Value* untereinander ausgibst. Und zwar so *0: Berlin* (Zeilenumbruch) *1: Wien* (Zeilenumbruch) *2: London* usw.

```
$stadt[] = "Berlin";
$stadt[] = "Wien";
$stadt[] = "London";
$stadt[] = "Moskau";
$stadt[] = "Paris";
```



3. Informiere dich im PHP-Handbuch über `foreach()` und schaue dir die ausführlichen Beispiele an.

4. Blättere zur Seite 32, zu unserem zweidimensionalen »Namens-Array«. Finde eine Lösung, das Array in folgender Weise auszugeben – dabei helfen dir die Recherchen von Aufgabe 3: Jeder »Datensatz« soll eine eigene Zeile bekommen und mit Bullets (Aufzählungszeichen) beginnen. Zwischen den einzelnen Einträgen einer Zeile genügt ein Leerzeichen. Nenne die Datei `aufgabe4.php`. So soll es also aussehen:

5. Wie muss die Auslesedatei `csv_ausgabe.php` von Seite 35 aussehen, wenn die `inhalt.csv` wie folgt verändert wird:

```
Überschrift a|Kurze Einleitungspassage|Textabsatz a
Überschrift b|Kurze Einleitungspassage|Textabsatz b
Überschrift c|Kurze, knappe Passage|Textabsatz c
```

6. (Zusatzaufgabe): Im Ordner `beispiele/kapitel01/guestbook` findest du ein einfaches Gästebuch mit Textdatei. Baue hier die Funktion `longkicker()` von Seite 28 ein. Überlange Kommentare sollen schon bei der Eingabe zwangsgetrennt werden. (Es genügt, wenn du die Funktion direkt in die PHP-Datei einbaust, eine extra Datei ist nicht nötig.)

Hinweis: Die Lösungen zu den Aufgaben findest du allesamt im Ordner `loesungen` auf der CD. Schaue in den Unterordner für das entsprechende Kapitel, im Beispiel in den Ordner `kapitel01`.