3 PHP

PHP ist eine Skriptsprache, die direkt in HTML-Seiten eingebettet wird, d.h. der Autor schreibt PHP-Befehle zusammen mit HTML-Befehlen in eine Datei. Wird diese Datei von einem Betrachter angefordert, so werden diese PHP-Befehle von einer *Zusatzsoftware* des Webservers Schritt für Schritt ausgeführt und die Ergebnisse an den Browser weitergeleitet.

Je nach Installation interpretiert diese PHP-Zusatzsoftware nur Dateien mit der Endung ".php".

PHP wird seit etwa 1994 entwickelt und erfreut sich stetig wachsender Beliebtheit. Ein besonderer Schwerpunkt liegt auf der Einbindung verschiedener Datenbanken. Die Sprache ist an C, Java und Perl angelehnt. Eine ausführliche Beschreibung liegt bei www.php.net.

3.1 PHP Befehle in HTML einbauen

Es gibt vier Möglichkeiten PHP-Code in HTML einzubetten:

```
<?php ... ?>
<? ... ?>
<script language="php"> ... </script>
<% ... %>
```

3.2 Datenausgabe mit echo und print

Die Befehle `echo` und `print` geben Texte (sogenannte Strings) aus:

```
<?php
   echo "Hallo Welt";
   print "Hallo Welt!";
?>
```

Der Unterschied zwischen echo und print besteht darin, dass echo auch mit mehreren, kommagetrennten Parametern umgehen kann. So ist echo beispielsweise in der Lage, einen String und einen Integer zusammen in einer einzigen Meldung auszugeben.

```
echo "Die Antwort ist ", 42;
// führt zur Ausgabe von: Die Antwort ist 42
```

Es gibt eine Reihe von Sonderzeichen. Wichtigstes Sonderzeichen: "\n": Neue Zeile. C- oder Perl-Kennern sind diese Zeichen natürlich vertraut.

```
echo "Hallo\n";
```

Man beachte dabei den Unterschied zu

```
echo "Hallo<BR>";
```

Im ersten Fall wird lediglich im vom Webserver ausgelieferten HTML-Text ein Zeilenumbruch durchgeführt. Im zweiten Fall enthält der HTML-Text den HTML-Befehl zum Zeilenumbruch.

PHP akzeptiert zwei Arten von Anführungszeichen, den doppelten und den einfachen. Oft ist es sinnvoll von beiden Gebrauch zu machen, was folgendes Beispiel zeigen soll:

```
echo "Hallo";

// ist generell gleichbedeutend mit

echo 'Hallo';

// Braucht man allerdings als Ausgabe die doppelten
// Anführungszeichen so kann man dies folgendermaßen
// erreichen:

echo 'Hallo, das hier sind doppelte Anführungszeichen: " ';

// oder umgekehrt:

echo "Hallo, das hier sind einfache Anführungszeichen: ' ";
```

Insbesondere bei der Ausgabe von HTML-Attributen ist es wichtig sich diese Funktionsweise zu Nutze zu machen.

```
echo "<table width="100%">";
// führt zu Fehlern

echo '<table width="100%">';
// erzeugt das gewünschte Ergebnis
```

Ist dieses Vorgehen nicht Möglich lassen sich Sonderzeichen mithilfe von s.g. Escape-Sequenzen markieren. Dies geschieht mit dem Voranstellen eines Backslashs.

```
echo "<table width=\"100%\">";
// erzeugt auch das gewünschte Ergebnis
```

3.3 Variablen

Um sinnvolle Anwendungen erstellen zu können, benötigen wir Variablen.

> Alle Variablennamen beginnen immer mit $

```
$text = "Ich bin ein String !";

echo $text;

$l = "langer";
$k = "kurzer";

echo "Ich bin ein $l$l$l$l$l$l Text!";
echo "Ich bin ein $k Text!";

$i = 10;
$j = 5;

echo $i,"+"$j,"=",$i+$j;
```

Der Typ der Variablen (ganze Zahl, Gleitkommazahl, String) wird je nach Verwendung von PHP automatisch bestimmt. Der Benutzer braucht sich darum nur in Spezialfällen kümmern.

3.4 Konstanten

Konstanten lassen sich in PHP wie folgt definieren:

```
define(”PI”,3.14159);

print PI;
// Ausgabe:  3.14159
```

Konstanten werden nicht mit einem vorangestellten $-Zeichen definiert, sobald sie einmal definiert wurden können sie nicht mehr geändert werden. Es gehört zum guten Programmierstil Konstanten ausschließlich mit Großbuchstaben zu definieren. PHP enthält bereits viele eingebaute Konstanten. Beispielsweise gehört zu den mathematischen Konstanten die Konstante M_PI:

```
print M_PI;
// Ausgabe:  3.14159265358979323846
```

Ob eine Variable definiert ist oder nicht, lässt sich mit einer einfachen if-Abfrage überprüfen:

```
...
if ( !defined('TEST') )
{
   echo 'Test ist nicht gesetzt!';
}
...
```

3.5 Arrays

PHP erzeugt Arrays mit dem Sprachkonstrukt array(). Im Folgenden werden einige Beispiele aufgezeigt, wie in PHP Arrays zu definieren sind.

```
$nummern = array( 1, 2, 3, 4, 5 );
$woerter = array( ”www”, ”Datenbank”, ”Anwendung” );


print $nummern[2];
// Zeigt das dritte Element des Arrays : 3

print $woerter[1];
// Zeigt das zweite Element des Arrays : Datenbank
```

Das erste Element in einem Array hat immer den Index 0. Die Werte des Arrays lassen sich unter Angabe des Indizes in eckigen Klammern abrufen und zuweisen.

```
$nummern[2] = 3;

$woerter[0] = ”web”;
```

3.5.1 Assoziative Arrays

Der Zugriff auf Array-Elemente ist bei assoziativen Arrays über String-Schlüssel möglich.

```
$liste = array( ”first”=>1, ”second”=>2, ”third”=>3 );

print $liste[”second”];
// Ausgabe:  2
```

3.5.2 Array-Elemente entfernen

Mit dem Aufruf von unset() können einzelne Array-Elemente oder sogar ganze Arrays löschen. Vorsicht! Löscht man ein Element aus einem Array werden die Indizes nicht Verschoben. Man löscht also auch den Index!

```
$liste = array( ”eins”, ”zwei”, ”drei” );

unset($liste([1]));

// Nun befinden sich nur noch zwei Elemente im Array:

print $liste[0];
// Ausgabe: eins
print $liste[2];
// Ausgabe: drei
// Das Element welches vorher den Index 1 hatte ist
// gelöscht worden
```

3.5.3 Reihenfolge im Array

Arrays ändern die Reihenfolge ihrer Einträge nicht, und neue Elemente werden bei einem vorhandenen Array hinten angefügt.

```
$liste = array( 1=>”eins”, 3=>”drei”, 5=>”fünf” );

$liste[2] = ”zwei”;
$liste[4] = ”vier”;
$liste[6] = ”sechs”;

// Diese Elemente werden in der Reihenfolge ihrer Definition
// hinten an das Array angehängt, sind über ihren Index jedoch
// eindeutig abrufbar
```

3.5.4 Mehrdimensionale Arrays

Es ist möglich mehrdimensionale Arrays zu definieren.

```
$werte = array(
               1=> array(1,2,3,4,5),
               2=> array(6,7,8,9,0),
               3=> array(”A”,”B”,”C”,”D”,”E”)
         );

// Abrufbar sind die Elemente nun folgendermaßen:

print $werte[1][2];
// Ausgabe: 3

print $werte[3][1];
// Ausgabe: B
```

3.5.5 Einige Grundfunktionen für Arrays

3.5.5.1 count

```
$werte = array( 2, 4, 6 );

print count( $werte );
// Ausgabe : 3
```

3.5.5.2 explode

Teilt einen String anhand eines Trennzeichens

```
array explode ( string separator, string string [, int limit] )
```

Gibt ein Array aus Strings zurück, die jeweils Teil von string sind. Die Abtrennung erfolgt dabei an der mit `separator` angegebenen Zeichenkette (ein oder mehrere Zeichen). Ist der Parameter Limit angegeben, werden maximal Limit Teile zurück

gegeben. Das letzte Element enthält dann den kompletten Rest von `string`.

Ist separator ein leerer String (""), so gibt `explode()` `FALSE` zurück. Enthält separator einen Wert, der nicht in string vorkommt, gibt `explode()` ein Array zurück, das den String als einziges Element enthält.

Ist der Parameter `limit` negativ, werden alle Teilstrings bis auf die letzten `limit` Teile zurückgegeben. Diese Funktionalität wurde in PHP 5.1.0 eingeführt.

Obgleich `implode()` aus historischen Gründen die Parameter in anderer Reihenfolge akzeptiert, verarbeitet `explode()` ausschließlich die hier angegebene. Stellen Sie daher sicher, dass Sie den Parameter `separator` vor dem Parameter `string` notieren.

```
// Beispiel 1

$pizza  = "Teil1 Teil2 Teil3 Teil4 Teil5 Teil6";
$teile = explode(" ", $pizza);
echo $teile[0]; // Teil1
echo $teile[1]; // Teil2

// Beispiel 2
$data = "foo:*:1023:1000::/home/foo:/bin/sh";
list($user, $pass, $uid, $gid, $gecos, $home, $shell) =
explode(":", $data);
echo $user; // foo
echo $pass; // *
```

3.5.5.3 implode

Verbindet Array-Elemente zu einem String

```
string implode ( string glue, array pieces )
```

Gibt einen String mit einer String-Repräsentation aller Array-Elemente in der gleichen Reihenfolge wie in dem Array zurück, bei dem die einzelnen Array-Elemente mit dem im Parameter glue angegebenen String verbunden werden.

```
$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);

print $comma_separated; // lastname,email,phone
```

3.6 Einfache Rechenoperationen

Folgende Rechenoperationen stehen in PHP zur Verfügung:

- "+": Addition, `$i+$j`,
- "-": Subtraktion, `$i-$j`
- "*": Multiplikation, `$i*$j`
- "/": Division, `$i/$j`
- "%": Restklasse

Dazu kommen noch ein paar Abkürzungen, um dem Programmierer das Leben zu erleichtern:

- `$i++` erhöht `$i` um 1.
- `++$i` erhöht `$i` ebenfalls um 1.
- `$i--` erniedrigt `$i` um 1.
- `--$i` erniedrigt `$i` ebenfalls um 1.

Der Unterschied zwischen `$i++` und `++$i` ist:

```
$i=0;
echo $i++;
```

gibt 0 aus, anschließend wird `$i` auf den Wert 1 erhöht.

```
$i=0;
echo ++$i;
```

erhöht zuerst `$i` auf 1 und gibt den Wert 1 aus.

Noch einige Beispiele:

```
$j = $j * 2;
// gleichbedeutend mit
$j *= 2;

$k = $k / 2;
// gleichbedeutend mit
$k /= 2;

u.s.w.
```

3.7 Strings aneinanderhängen

Beliebige Zeichenketten lassen sich durch den Punkt-Operator aneinander hängen. Hierbei ist es egal ob Sie Strings und Zahlen mischen.

```
// Beispiel: Strings verbinden mit dem Punkt

$l = "langer";
$k = "kurzer";
echo $l . ' ' . $k;
// ergibt: langer kurzer

$l .= $k;
echo $l;
// ergibt: langerkurzer

$zahl = 5;
$j = 'Hallo Herr '.$l.'. Sie sind der '.$zahl.'te Besucher.';
echo $j;

// ergibt: Hallo Herr langer. Sie sind der 5te Besucher.
```

3.8 Schleifen in PHP

3.8.1 while-Schleifen

Gleich ein Beispiel:

```
$t = "Ich soll meine Uebungsaufgaben selbst erstellen!<BR>\n";
$i = 0;

while ($i<10) {
   echo $t;
   $i++;
}
```

Hier wird 10-mal der Text in der Variablen `$t` ausgegeben. Zu Beginn wird `$i` auf 0 gesetzt. `$i` wird in jedem Schleifendurchlauf um 1 erhöht, bis `$i` den Wert 10 erreicht. Dann ist die Bedingung `$i<10` nicht mehr wahr und die Schleife bricht ab.

3.8.2 do-while-Schleife

```
$t = "Ich soll meine Uebungsaufgaben selbst lösen!<BR>\n";
$i = 0;

do {
   echo $t;
   $i++;
} while ($i<10);
```

Wo ist der Unterschied?

Man bemerkt den Unterschied, wenn z.B. statt `$i=0;` zu Beginn `$i=10;` gesetzt wird. Im ersten Fall ist die Bedingung `$i<10` nicht wahr und die Befehle innerhalb der geschweiften Klammern werden nicht ausgeführt.

Im zweiten Fall werden zuerst die Befehle innerhalb der geschweiften Klammern ausgeführt, danach wird getestet, ob `$i<10`. Dies ist nicht der Fall, also wird abgebrochen. D.h. aber, die Schleife wird **mindestens einmal** durchlaufen.

3.8.3 for-Schleife

Eine weitere Möglichkeit, eine Schleife zu programmieren, ist der `for`-Befehl. Gleich ein Beispiel:

```
$t = "Ich soll meine Uebungsaufgaben selbst lösen!<BR>\n";

for ($i=0;$i<10;$i++) {
   echo $t;
}
```

Der `for`-Befehl besteht aus drei Ausdrücken.

```
for (ausdruck1;ausdruck2;ausdruck3) { ... }
```

- Mit `ausdruck1` wird die Schleife initialisiert, d.h. normalerweise wird die Variable, die die Schleifendurchläufe zählt, auf den Anfangswert gesetzt.
- `ausdruck2` gibt die Abbruchbedingung an.
- In `ausdruck3` wird die Variable, die die Schleifendurchläufe zählt, erhöht bzw. erniedrigt.

Der `for`-Befehl hat den Vorteil, daß alle zur Kontrolle der Schleife nötigen Befehle in einer Zeile stehen. Ein weiteres Beispiel, diesmal wird heruntergezählt.

```
$t = "Ich soll meine Uebungsaufgaben selbst lösen!<BR>\n";

for ($i=10;$i>0;$i--) {
   echo $t;
}
```

3.9 Bedingungen und Verzweigungen

3.9.1 if-else-Anweisung

Zur Fallunterscheidung gibt es den `if`-Befehl:

```
if ($i<0) {
   echo "$i ist kleiner als Null\n";
}
```

oder auch

```
if ($i<0) {
   echo "$i ist kleiner als Null\n";
} else {
   echo "$i ist nicht kleiner als Null\n";
}
```

Man kann diesen Befehl auch schachteln:

```
if ($i<0) {
   echo "$i ist kleiner als Null\n";
} else if ($i>0) {
   echo "$i ist groesser als Null\n";
} else {
   echo "$i ist gleich Null\n";
}
```

3.9.2 switch-case

Hat man mehrere Tests der gleichen Variable, so kann man mit dem `switch`-Befehl Arbeit einsparen:

```
switch ($name) {
   case "Heinrich":
         echo "Ich bin der kluge Heinrich";
   break;
   case "Hans":
         echo "Ich bin der dumme Hans";
   break;
   case "Agathe":
         echo "Ich bin die Agathe";
   break;
   default:
         echo "Wir sind der Rest";
}
```

Falls die Variable `$name` den Wert `"Hans"` hat, wird als nächster Befehl

```
echo "Ich bin der dumme Hans";
```

ausgeführt. Normalerweise würden dann alle nachfolgenden Befehle ausgeführt werden, u.a.

```
echo "Ich bin die Agathe";
```

Dies ist meist nicht erwünscht, man springt deshalb mit `break` aus dem `switch`-Befehl heraus.

3.10 Vergleichsoperatoren

Bisher konnten wir testen, ob `$i<10` oder `$i>10` gilt. Es gibt aber noch mehr Möglichkeiten:

- `$i==10`: Ist `$i` gleich 10?
- `$i===10`: Ist `$i` gleich 10 und sind die Variablentypen exakt gleich?
- `$i!=10`: Ist `$i` ungleich 10?
- `$i>=10`: Ist `$i` größer oder gleich 10?
- `$i<=10`: Ist `$i` kleiner oder gleich 10?

Man kann auch kombinieren:

- `($i==10) && ($j>0)`: Ist `$i` gleich 10 **UND** `$j` größer als 0?
- `($i==10) || ($j==0)`: Ist `$i` gleich 10 **ODER** `$j` gleich 0?

```
// Beispiel:

$i = '10'; // wird als String abgelegt

if( $i == 10 ) // true
{
    ...
}

if( $i === 10 ) // false
{
    ...
}
```

3.11 Dokumente und Inhalte einfügen

include()

Mit `include()` lassen sich an beliebiger Stelle im PHP-Quelltext andere Dateien einbinden. Der Inhalt der eingefügten Datei wird anstelle des Befehls in den Quelltext gesetzt. Falls die eingefügte Datei eine PHP-Datei ist, so wird dieser Programmcode beim Parsen berücksichtigt. Sollte das Einfügen der geforderten Datei nicht funktionieren (bsw. weil die Datei nicht vorhanden ist) wird eine Warnmeldung (warning) ausgegeben. Das Skript wird jedoch weiter ausgeführt.

```
<?php
   // PHP-Code
   // ...
   include('menu.php');
   // ...
   // PHP-Code
?>
```

require()

`require()` funktioniert analog zu `include()`. Der einzige Unterschied liegt in der Fehlerbehandlung. Im Falle eines Fehlers liefert `require()` eine Fehlermeldung (fatal error) zurück und das PHP-Skript wird an dieser Stelle unterbrochen.

include_once() und require_once()

Um sicher zu stellen, dass eine Datei nur einmal eingebunden wird kann man `include_once()` oder `require_once()` verwenden. Diese Funktionen verhindern das mehrfache Einfügen desselben Dokuments innerhalb eines Programms. Ansonsten verhalten sich die Funktionen analog zu `include()` bzw. `require()`.

Aus Sicherheitsgründen ist es absolut erforderlich eine Variable, welche einen Dateinamen enthält, vor der Übergabe an `include()` oder `require()` zu prüfen. Ansonsten kann es einem Angreifer möglich sein fast beliebige Dateien des Webservers auszulesen.

3.12 Variableninhalte an andere Dokumente senden

Will man Inhalte von Variablen anderen PHP-Dokumenten zwecks einer Weiterverarbeitung zur Verfügung stellen, muss man sich Gedanken machen, wie diese Daten dort hin transportiert werden sollen. Generell stehen drei verschiedene Varianten zur Verfügung.

- `$_GET` – Übergabe erfolgt über die Adresszeile des Webbrowsers
  - Eignet sich sehr gut für die Steuerung der Seite (Komponentenwahl, Seitennummer, etc.)
  - Alle Übergaben stehen im Klartext in der Adresszeile des Browsers (ermöglicht das Versenden von direkten Links auf eine bestimmte Seite (Komonente), das Übermitteln brisanter Daten im Klartext kann bedenklich sein)
- `$_POST` – Übergabe erfolgt innerhalb der HTTP-Message im Datenblock
  - Eignet sich sehr gut für das Übergeben von Formulardaten
  - Eignet sich weniger gut für die Steuerung der Seite
  - Für einen sicheren Transport der Daten muss auf Verschlüsselungstechniken zurückgegriffen werden (z.B. SSL)
- `$_GLOBALS` – Übergabe mittels globaler Variablen
  - Wird aus Sicherheitsgründen nicht von jedem Webserver unterstützt
  - Es kann zu Namenskonflikten kommen

Wir werden uns im Folgenden ausschließlich mit den Methoten `$_GET` und `$_POST` auseinandersetzen. Das Verwenden globaler Variablen ist meist nicht zu empfehlen. Nun soll anhand von HTML-Formularen das Verwenden dieser beiden Methoden erklärt werden:

3.12.1 Formulare mit PHP

Die Darstellung von Formularen erfolgt durch normales HTML. Die Versendung der Daten übernimmt das Hypertext Transfer Protocol (HTTP). Es gibt zwei Methoden, mit denen Formulardaten übertragen werden. Diese Methode wird im Form-Tag <form> des Formulars festgelegt. Die eine Methode ist `$_POST`, die andere `$_GET`. Wenn sie einmal ein Formular mit der einen und der anderen Methode festlegen und dann an eine beliebige Datei weiterschicken (action=""), so werden sie den Unterschied in der Adresszeile Ihres Browsers sehen können. Die Methode `$_GET` schickt die Daten als Querystring mit. Der Querystring wird mittels Fragezeichen '?' mit dem Dateinamen bzw. der Adresse verknüpft. Jedes Formularelement, das mittels HTML-Attribut name sauber benannt wurde, wird hier mit name=wert überreicht. Das sieht dann so aus:

```
http://localhost/test.php?name=wert&andererName=andererWert
```

Die Länge der URL ist durch die Browser begrenzt. Die Datenmenge, die so übermittelt werden kann, ist dadurch auf etwa 2 Kb beschränkt. Alle Daten sind für den Benutzer in der Adresszeile des Browsers sichtbar.

Die Methode post legt die Daten innerhalb der Nachricht ab und versendet sie verpackt. Der Platz unterliegt keiner Beschränkung. Die Daten sind zumindest nicht offen sichtbar.

Die Entgegennahme der Daten mit PHP ist denkbar einfach: Jedes Formularelement verfügt über das Attribut name=' '. Diesen Namen erkennt PHP als Schlüssel des globalen Arrays $_POST. Falls Sie als Methode für das Formular get angegeben haben, werden die Wert im Array $_GET gespeichert. Im oben angegebenen Beispiel kann der Inhalt des Textfeldes so mit der Variable `$_POST["IhrName"]` weiterverarbeitet werden. Beachten Sie die Groß- und Kleinschreibung.

Manchmal ist es aus konzeptionellen Gründen erwünscht, dass sich beide Teile, Formular und weiterverarbeitendes Skript, innerhalb einer einzigen Datei befinden. In diesem Fall muss die in action angegebene Datei die aufrufende Datei selber sein.

Aus Sicherheitsgründen ist es unbedingt erforderlich alle Übergaben auf Korrektheit zu prüfen. Generell darf allem, was per $_GET, $_POST oder $_GLOBALS übergeben wird **nicht** vertraut werden! Alle diese Variablen lassen sich von einem Angreifer beliebig manipulieren! Überlegen Sie was für Möglichkeiten ein Angreifer bei folgendem Code hätte:

```
...
include($_GET['component']);
...
```

Ein Angreifer könnte nun beispielsweise die folgende Adresszeile erzeugen:

```
.../index.php?component=/etc/passwd
```

Dies hätte die Anzeige der Datei /etc/passwd im Browser eines Angreifers zur Folge!

Beispiel für das Verwenden von $_POST:

```
// formular.php
...
<?php
if ($_POST["send"]) {
   print $_POST["IhrName"];
}
else {
?>

<form action="formular.php" method="post">
   <input type="text" name="IhrName">
   <input type="submit" name="send" value="Abschicken">
</form>

<?
}
?>
...
```

Beispiel für das Verwenden von $_GET:

```
// Adresszeile des Browsers:
// http://localhost/test.php?com=news&newsid=14

...
<?php
   print $_GET["com"];
   // Ausgabe: news
   print $_GET["newsid"];
   // Ausgabe: 14
?>
...
```

Beispiel: Wechsel der Hintergrundfarbe:

```
...
<?php
$bg = $_POST["bg"];
switch ($bg) {
   case "rot":
      $bgcolor = "#FF0000";
      break;
   case "grün":
      $bgcolor = "#00FF00";
      break;
   case "blau":
      $bgcolor = "#0000FF";
      break;
   default:
      $bgcolor = "#FFFFFF";
}
?>
<body bgcolor="<? print $bgcolor; ?>">

<form action="<? print $PHP_SELF; ?>" method="post">
<input type="submit" name="bg" value="rot">
<input type="submit" name="bg" value="grün">
<input type="submit" name="bg" value="blau">
</form>
</body>
...
```

Beispiel: Anlegen eines Textfeldes:

```
...
<?php
if ($_POST["send"]) {
   print($_POST["IhreNachricht"]);
}
else {
   ?>
         <form action="formular.php" method="post">
         <textarea name="IhreNachricht"></textarea>
         <input type="submit" name="send" value="Abschicken">
         </form>
   <?php
}
?>
...
```

3.13 Anwendungsdesign mit Zentraldokument

Im Verlauf einer Entwicklung werden die Anforderungen an ein Anwendungssystem meist immer komplexer. Eine Anwendung muss in der Lage sein vielfältige Aufgaben zu erfüllen und diese den Benutzern der Anwendung möglichst weitgehend an deren Bedürfnisse anpassbar aufzubereiten. Gleichzeitig muss die Anwendung eine restriktive Benutzer- und Rechtesteuerung realisieren.

Ein Ansatz wäre es sicher alle diese Eigenschaften in jeder PHP-Datei erneut abzufragen. Dies hätte jedoch zur Folge, dass enorm viel redundanter Programmcode in der Anwendung gepflegt werden müsste. Dieser hohe Aufwand hat meist zur Folge, dass Sicherheit und Stabilität der Anwendung auf der Strecke bleibt. Aus diesen und vielen weiteren Gründen macht es Sinn eine zentrale Datei zu erstellen, die alle oben beschriebenen Aufgaben übernimmt. Sämtliche Seitenaufrufe müssen nun jedoch zwangsweise über dieses zentrale Dokument geleitet werden. Außerdem muss es wirkungsvoll unterbunden werden, dass auf alle anderen Dokumente direkt (außerhalb des zentralen Dokuments) zugegriffen werden kann. Das Zentraldokument übernimmt also die Steuerung, wann welches Dokument an welcher Stelle auf der Seite angezeigt wird.

Eigenschaften einer Anwendung mit Zentraldokument:

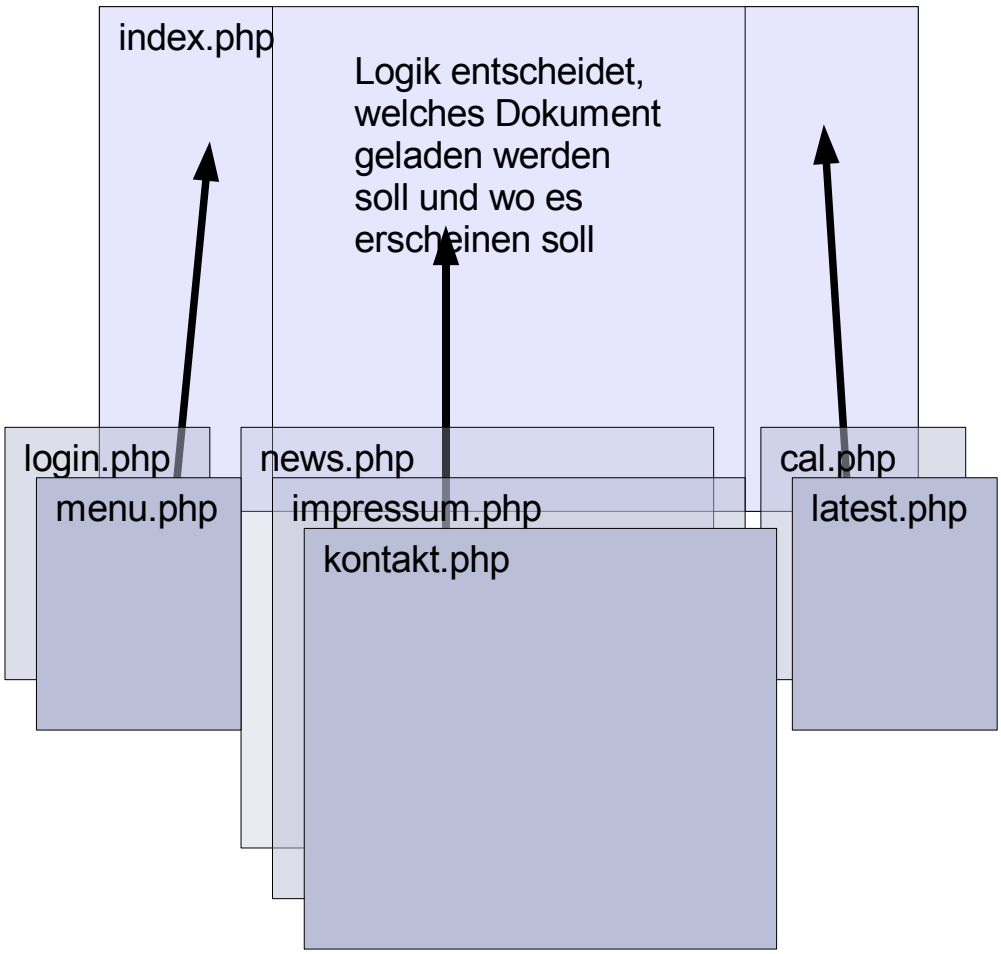
- Jeder Seitenaufruf durchläuft zwangsweise die index.php
- Direkter Aufruf von Unterseiten wird untersagt

Folgende Aufgaben kann ein Zentraldokument u.a. übernehmen:

- Steuerungsfunktionen, Einbinden der richtigen Komponenten

- Generieren des Gerüstes der Seite
- Rechteverwaltung der Seite
- Sessionsteuerung
- Schutz aller Komponenten
- ...

Beispielstruktur einer Anwendung mit Zentraldokument:



Die Komponenten (Dokumente) der Anwendung werden für jeden Besucher der Seite, je nachdem welche Rechte dieser hat und welche Komponente er anfordert, in das Zentraldokument per `require()` eingebunden. Die Entscheidung wann und wo eine Komponente geladen werden soll hängt einerseits von den per `$_GET` übermittelten Daten und andererseits von evtl. innerhalb einer Datenbank gespeicherter Informationen ab.

Beispielcode eines sehr einfachen Zentraldokuments (index.php):

```
<?php
   // Sessionsteuerung etc.

   // Wir sind innerhalb der Datei index.php
   define('INDEX',true);

   // Auswerten der vom Benutzer gewünschten Komponente
   if( isset($_GET['com']) )
   {
         // Prüfen ob die gewünschte Komponente erlaubt ist,
         // Dies kann mittels einer Datenbankabfrage oder
         // auf jede andere Art erfolgen. WICHTIG ist aber
         // das geprüft wird!
         ...
         $com['CENTER'] = $_GET['com'];
   }
   else
   {
         // Lade eine Standardkomponente
         $com['CENTER'] = 'news';
   }

   // Bestimmen aller anderen Komponenten:
   // Die Funktion getComponents gibt in unserem Beispiel
   // ein Array mit allen für diesen Benutzer und diese
   // Hauptkomponente passenden Komponenten zurück
   $com['LEFT'] = getComponents('LEFT');
   $com['RIGHT'] = getComponents('RIGHT');

   // Erstellen des Gerüsts der Seite:
   echo '<table>';
   echo '<tr>';
   echo '<td>';
```

```
// Einbinden aller Komponenten an der richtigen Stelle
for($i=0;$i<count($com['LEFT']);$i++)
{
      ...
      require($com['LEFT'][$i].'.php');
      ...
}
echo '</td>';
echo '</tr>';
echo '<tr>';
echo '<td>';
// Einbinden der zentralen Komponente
require($com['CENTER'].'.php');
...
```

Das obige Beispiel lässt sich nicht direkt per copy and paste übernehmen, da die Logik der Funktion `getComponents()` nicht bekannt ist. Hier wird einfach davon ausgegangen, dass diese Funktion alle Komponenten, welche für den ihr übergebenen Bereich in Form eines assoziativen Arrays zurückliefert. Im Regelfall wird die Entscheidung welche Komponenten dies sind auf einigen Datenbankaufrufen basieren.

Da wir in unserem Zentraldokument eine Variable INDEX definiert haben und ihr den Wert `true` zugewiesen haben, können wir diese Variable innerhalb aller Unterdokumente abfragen. Die Bearbeitung des Unterdokuments wird ausschließlich dann erlaubt, wenn diese Variable auf den Wert `true` definiert wurde. Diese Methode bietet für die meisten Fälle einen recht guten Schutz. Es sei jedoch darauf hingewiesen, dass evtl. bessere Schutzmaßnahmen erforderlich sind (z.B. Sessions, etc).

3.14 Operationen auf Strings

3.14.1 strcmp

Binärer Vergleich zweier Strings

```
int strcmp ( string str1, string str2 )
```

Ist `str1` kleiner als `str2` wird < 0, ist `str1` größer als `str2` wird > 0 und bei Gleichheit wird Null zurück gegeben.

Beachten sie, dass zwischen Groß- und Kleinschreibung unterschieden wird.

3.14.2 strncmp

Vergleich von n Buchstaben

```
int strncmp ( string str1, string str2, int len )
```

Diese Funktion ist ähnlich `strcmp()` nur mit dem Unterschied, dass sie den maximalen Wert der Zeichen (`len`) angeben können, der verglichen werden soll. Ist einer der Strings kürzer als (`len`) wird dessen Länge für den Vergleich angewendet.

Die Funktion gibt einen Wert kleiner Null zurück, wenn `str1` kleiner ist als `str2` bzw. > 0, wenn `str1` größer ist als `str2` oder Null, wenn sie gleich sind.

Dieser Vergleich unterscheidet Klein- und Großschreibung.

3.14.3 strlen

Ermitteln der String-Länge

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

Gibt die Länge der Zeichenkette string zurück.

3.14.4 strpos

Sucht erstes Vorkommen des Suchstrings und liefert die Position

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

Gibt als numerischen Wert die Position des ersten Vorkommens von `needle` innerhalb der Zeichenkette `haystack` zurück. Anders als bei `strrpos()` kann diese Funktion eine komplette Zeichenkette unterstützen.

Wurde `needle` nicht gefunden, wird `FALSE` zurück gegeben.

Anmerkung: Die Meldungen "Zeichen bei Position 0 gefunden (character found at position 0)" und "Zeichen nicht gefunden (character not found)" werden oft falsch ausgelegt. Hier die Unterscheidung:

```
$string1 = ”Hallo”;
$string2 = ”lo”;
echo strpos($string1,$string2);

// Ausgabe : 3
```

Der optionale Parameter `offset` ermöglicht es, den Startwert für die Suche nach `needle` innerhalb von `haystack` anzugeben. Die zurück gegebene Positions-Angabe ist dann relativ zum Anfang von `haystack`.

3.14.5 strrpos

Sucht letztes Vorkommen des gesuchten Zeichens und liefert die Position

```
int strrpos ( string haystack, char needle )
```

Der Wert des letzen Vorkommens von `needle` im String `haystack` wird ermittelt. `Needle` darf nur ein einzelnes Zeichen sein. Wird als `needle` eine Zeichenkette übergeben, wird nur dessen erster Buchstabe ausgewertet.

Wurde `needle` nicht gefunden, wird `FALSE` zurück gegeben.

Ist `needle` kein String, wird er zu einem Integer-Wert umgesetzt und als das diesem Wert entsprechende ASCII-Zeichen angesehen.

3.14.6 strchr

Sucht erstes Vorkommen des gesuchten Zeichens und liefert den Reststring

```
string strchr ( string haystack, string needle )
```

Diese Funktion ist ein Alias zu `strstr()` und funktioniert genau wie dort beschrieben.

3.14.7 strrchr

Sucht letztes Vorkommen des gesuchten Zeichens und liefert den Reststring

```
string strrchr ( string haystack, string needle )
```

Diese Funktion gibt den letzten Teil des Strings `haystack` zurück, der ab dem letzten Vorkommen von `needle` steht (bis zum Ende).

Wird `needle` nicht gefunden, wird `FALSE` zurück gegeben.

Enthält `needle` mehr als ein Zeichen, wird nur das erste Zeichen genommen.

Ist `needle` kein String, wird er zu einem Integer-Wert umgesetzt und als das diesem Wert entsprechende ASCII-Zeichen angesehen.

```
$string1 = ”Hallo”;
$string2 = ”l”;
echo strrchr($string1,$string2);

// Ausgabe : lo
```

3.14.8 strstr

Sucht erstes Vorkommen des Suchstrings und liefert den Reststring

```
string strstr ( string haystack, string needle )
```

Gibt den String `haystack` ab dem ersten Vorkommen von `needle` bis zum Ende zurück.

Falls `needle` nicht gefunden wird, ist das Ergebnis FALSE.

Ist `needle` kein String, wird er zu einem Integer-Wert umgesetzt und als das diesem Wert entsprechende ASCII-Zeichen angesehen.

Anmerkung: Diese Funktion unterscheidet zwischen Groß- und Kleinschreibung. Ist keine Unterscheidung gewünscht / erforderlich, sollten Sie `stristr()` verwenden.

```
Beispiel 1. strstr() Beispiel
$email = 'user@example.com';
$domain = strstr($email, '@');
print $domain; // Ausgabe: @example.com
```

Anmerkung: Wenn Sie nur herausfinden möchten, ob ein bestimmter needle innerhalb von haystack vorkommt, verwenden Sie stattdessen die schnellere und weniger speicherintesive Funktion strpos().

3.14.9 strtok

Zerlegt einen String

```
string strtok ( string str, string token )
```

strtok() zerlegt einen String (str) in kürzere Strings (Tokens), wobei jeder Token von im Parameter token festgelegten Zeichen begrenzt wird. Das bedeutet, dass, wenn Sie einen String wie "Dies ist ein Beispiel-String" haben, Sie ihn in seine einzelnen Worte zerlegen können, wenn Sie das Leerzeichen als Token verwenden.

```
Beispiel 1. strtok() Beispiel
<?php
$string = "Dies ist\tein Beispiel-\nString";
/* Sowohl das Tabulator- als auch das Newline-Zeichen werden
   zusätzlich zum Leerzeichen als Token zum Zerlegen verwendet
*/
$tok = strtok($string, " \n\t");

while ($tok != false) {
    echo "Wort=$tok<br />";
    $tok = strtok(" \n\t");
}
?>
```

Beachten Sie, dass nur der erste Aufruf von strtok das String-Argument verwendet. Jeder Folgeaufruf von strtok benötigt nur die zu beachtenden Token, um die für den aktuellen String notwendigen herauszufinden. Um neu zu beginnen oder einen neuen String zu zerlegen, müssen Sie nur erneut strtok mit dem string-Parameter aufrufen,

damit die Funktion neu initialisiert wird. Beachten Sie, dass Sie mehrere Tokens im Token-Parameter angeben können. Der String wird dann an jeder Stelle zerlegt, an der eines der angegebenen Zeichen gefunden wird.

Das Verhalten der Funktion beim Auffinden eines leeren Teils veränderte sich mit PHP Version 4.1.0. Zuvor wurde ein leerer String zurückgegeben, wohingegen das neue, korrekte Verhalten diesen Teil des Strings verwirft.

3.14.10 strtolower

Setzt einen String in Kleinbuchstaben um

```
string strtolower ( string str )
```

Gibt string zurück, in dem alle Buchstaben in Kleinbuchstaben umgewandelt wurden.

```
Beispiel 1. strtolower() Beispiel
<?php
$str = "Mary Hat Ein Kleines Lamm, und Sie LIEBT Es So.";
$str = strtolower($str);
echo $str; // Gibt aus: mary hat ein kleines lamm, und sie
liebt es so.
?>
```

3.14.11 strtoupper

Setzt einen String in Großbuchstaben um

```
string strtoupper ( string string )
```

Gibt string zurück, in dem alle Buchstaben in Großbuchstaben umgewandelt wurden.

```
Beispiel 1. strtoupper() Beispiel
<?php
$str = "Mary Hat Ein Kleines Lamm, und Sie LIEBT Es So.";
```

```
$str = strtoupper($str);
echo $str; // Gibt aus: MARY HAT EIN KLEINES LAMM, UND SIE
LIEBT ES SO.
?>
```

3.14.12 strtr

Tauscht bestimmte Zeichen aus

```
string strtr ( string str, string from, string to )
```

```
string strtr ( string str, array replace_pairs )
```

Diese Funktion gibt eine Kopie von str zurück, in der alle Vorkommen jedes Zeichens von from in das korrespondierende Zeichen in to umgewandelt wurde.

Haben from und to eine unterschiedliche Länge, werden die überzähligen Zeichen im jeweils längeren Parameter ignoriert.

```
Beispiel 1. strtr() Beispiel
<?php
$addr = strtr($addr, "äåö", "aao");
?>
```

strtr() kann auch mit nur zwei Argumenten aufgerufen werden. Wenn der Aufruf mit zwei Argumenten durchgeführt wird, verhält sich die Funktion anders: from muss nun ein Array sein, das string->string-Paare enthält, die im Originalstring ersetzt werden sollen. strtr() tauscht dabei zuerst die längsten möglichen Treffer aus und verändert

bereits durchgeführte Ersetzungen *NICHT*.

```
Beispiel 2. strtr() Beispiel mit zwei Argumenten
<?php
$trans = array("hallo" => "hi", "hi" => "hallo");
echo strtr("hi ihr, ich sagte hallo", $trans);
?>

// Ausgabe: hallo ihr, ich sagte hi
```

3.14.13 substr

Gibt einen Teil eines Strings zurück

```
string substr ( string string, int start [, int length] )
```

substr() gibt den Teil von string zurück, der durch die Parameter start und length definiert wurde.

Wenn start positiv ist, beginnt der zurück gegebene String an der start-Position von string, angefangen bei 0 (Null). So ist z.B. im String 'abcdef' das Zeichen an der Position 0 gleich 'a', das Zeichen an der Position 2 ist 'c' usw.

```
Beispiel 1. Generelle Verwendung von substr()
<?php
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
echo substr('abcdef', 0, 8);  // abcdef
echo substr('abcdef', -1, 1); // f
```

```
// Auf ein einzelnes Zeichen eines Strings kann auch mittels
// geschweifter Klammern erfolgen
$string = 'abcdef';
echo $string{0};                 // a
echo $string{3};                 // d
echo $string{strlen($string)-1}; // f

?>
```

3.14.14 trim

Entfernt Whitespaces (oder andere Zeichen) am Anfang und Ende eines Strings

```
string trim ( string str [, string charlist] )
```

Die Funktion entfernt Whitespaces an Anfang und Ende von str und gibt den String dann zurück. Ohne Verwendung des zweiten Parameters entfernt trim() folgende Zeichen:

```
" "    (ASCII 32 (0x20)), ein normales Leerzeichen.
"\t"   (ASCII 9 (0x09)), ein Tabulatorzeichen.
"\n"   (ASCII 10 (0x0A)), einen Zeilenvorschub (Line Feed).
"\r"   (ASCII 13 (0x0D)), ein Wagenrücklaufzeichen
       (Carriage Return).
"\0"   (ASCII 0 (0x00)), das NUL-Byte.
"\x0B" (ASCII 11 (0x0B)), ein vertikaler Tabulator.
```

Optional kann eine Liste weiterer Zeichen angefügt werden, die an Anfang und Ende

der Zeichenkette entfernt werden sollen. Um diese Zeichen anzugeben, wird der charlist Parameter verwendet. Er enthält eine Liste aller zu entfernenden Zeichen. Mit .. können darüber hinaus auch ganze Bereiche von Zeichen angegeben werden.

```
Beispiel 1. Beispiel zur Verwendung von trim()
<?php

$text = "\t\tDieser Text besteht aus mehreren Wörtern :) ...
";

echo trim($text);           // "Dieser Text besteht aus
mehreren Wörtern :) ..."
echo trim($text, " \t.");   // "Dieser Text besteht aus
mehreren Wörtern :)"

// Trimmen der ASCII Steuerzeichen an Anfang und Ende von
$binary
// (inklusive der Zeichen von ASCII 0 bis 31)
$clean = trim($binary, "\x00..\x1F");
?>
```

3.14.15 str_pad

str_pad - Erweitert einen String auf eine bestimmte Länge unter Verwendung eines anderen Strings

```
string str_pad ( string input, int pad_length [, string
pad_string [, int pad_type]] )
```

Mit dieser Funktion erweitern Sie den String input auf der linken, rechten oder beiden Seiten auf die mit dem optionalen Parameter pad_length angegebene Länge. Wird das optionale Argument pad_string nicht angegeben, erfolgt die Erweiterung mit Leerzeichen, ansonsten wird hierzu der Inhalt von pad_string verwendet.

Das optionale Argument pad_type kann STR_PAD_RIGHT, STR_PAD_LEFT oder STR_PAD_BOTH sein. Wird pad_type nicht angegeben, so wird per default von STR_PAD_RIGHT ausgegangen.

Ist der Wert von pad_length negativ oder kleiner als die Länge des Input-Strings, wird keine Erweiterung vorgenommen.

```
Beispiel 1. str_pad()-Beispiel:
$input = "Alien";
print str_pad ($input, 10);                      // Ergebnis
"Alien     "
print str_pad ($input, 10, "-=", STR_PAD_LEFT); // Ergebnis
"-=-=-Alien"
print str_pad ($input, 10, "_", STR_PAD_BOTH);  // Ergebnis
"__Alien___"
```

3.14.16 str_word_count

str_word_count – Gibt Informationen über Worte in einem String zurück

```
mixed str_word_count ( string string [, int format] )
```

Zählt die Worte in string. Wenn der optionale Parameter format nicht angegeben wird, wird ein Integer mit der Anzahl der Worte zurückgegeben. Andernfalls wird ein Array zurückgegeben, dessen Inhalt von format abhängt. format kann folgende Werte annehmen.

1 – gibt einen Array zurück, der alle Worte innerhalb von string enthält.

2 – gibt einen asoziativen Array zurück, dessen Schlüssel die Position des Wortes innerhalb von string ist und dessen Wert das eigentliche Wort ist.

3.14.17 str_replace

Ersetzt alle Vorkommen eines Strings in einem anderen String

```
mixed str_replace ( mixed search, mixed replace, mixed
subject )
```

Diese Funktion ersetzt alle Vorkommen von search innerhalb der Zeichenkette subject durch den String replace. Falls Sie keine ausgefallenen String-Ersetzungen brauchen, sollten Sie immer dieser Funktion den Vorzug vor ereg_replace() oder preg_replace() geben.

Ist subject ein Array, erfolgt das Suchen und Ersetzen an jedem Wert von subject, und der Rückgabewert ist ebenfalls ein Array.

Sind search und replace Arrays, nimmt str_replace() einen Wert von jedem Array und verwendet ihn zum Suchen und Ersetzen von subject. Hat replace weniger Werte als search, so wird ein leerer String für den Rest der Werte zum Ersetzen verwendet. Ist search ein Array und replace ein String, dann wird dieser String für jeden Wert von search angewandt.

```
echo str_replace("euch","dir","Hallo wie geht es euch?");
// Ausgabe : Hallo wie geht es dir?

$search = array("Hallo","euch");
$replace = array("Guten Tag","dir");

echo str_replace($search,$replace,"Hallo wie geht es euch?");
// Ausgabe : Guten Tag wie geht es dir?
```

3.14.18 str_repeat

Wiederholt eine String-Ausgabe

```
string str_repeat ( string input, int multiplier )
```

Gibt input_str multiplier mal zurück, wobei multiplier größer als 0 sein muss.

```
Beispiel 1. str_repeat()-Beispiel:
echo str_repeat ("-=", 10);
```

Es wird "-=-=-=-=-=-=-=-=-=-=" ausgegeben.

3.14.19 ucfirst

Verwandelt das erste Zeichen eines Strings in einen Großbuchstaben

```
string ucfirst ( string str )
```

Wandelt das erste Zeichen von str in einen Großbuchstaben um, wenn es ein Zeichen des Alphabets ist, und gibt den veränderten String zurück.

Beachten Sie, dass die Zeichen des Alphabets abhängig sind vom Wert des gesetzten

locale erkannt werden. Ist der Voreinstellung "C" werden Sonderzeichen wie die deutschen Umlaute (ä etc.) nicht erkannt und daher nicht umgewandelt.

```
Beispiel 1. ucfirst()-Beispiel
<?php
$foo = 'hallo welt!';
$foo = ucfirst($foo);             // Hallo welt!

$bar = 'HALLO WELT!';
$bar = ucfirst($bar);             // HALLO WELT!
$bar = ucfirst(strtolower($bar)); // Hallo welt!
?>
```

3.14.20 ucwords

Wandelt jeden ersten Buchstaben eines Wortes innerhalb eines Strings in einen Großbuchstaben

```
string ucwords ( string str )
```

Gibt einen String zurück, in dem das erste Zeichen eines jeden Wortes innerhalb von str in einen Großbuchstaben umgewandelt wurde, sofern es sich dabei um Buchstaben handelt.

Als Wort wird hierbei eine Zeichenkette verstanden, die einem Whitespace (Leerzeichen, Seitenvorschub, Zeilenvorschub, Wagenrücklauf sowie horizontalem und vertikalem Tabulatorzeichen) folgt.

```
Beispiel 1. ucwords()-Beispiel
<?php
$foo = 'hallo welt!';
$foo = ucwords ($foo);          // Hallo Welt!

$bar = 'HALLO WELT!';
$bar = ucwords($bar);             // HALLO WELT!
$bar = ucwords(strtolower($bar)); // Hallo Welt!
?>
```

3.15 Reguläre Ausdrücke

Reguläre Ausdrücke können für komplexe Manipulationen an Zeichenketten verwendet werden. PHP bietet für unterschiedliche Aufgabenfälle verschiedene Funktionen an. Im Folgenden sollen einige dieser Funktionen erläutert werden.

3.15.1 ereg

Die Funktion ereg() (eregi() ohne Berücksichtigung von Groß- und Kleinschreibung) gibt 'true' zurück, falls der reguläre Ausdruck innerhalb der Zeichenkette gefunden wurde.

```
<?php
   $string = 'Hallo';
   ereg('lo',$string); // liefert 'true'
   ereg('^H',$string); // liefert 'true'
   ereg('^[0-9]',$string); // liefert 'false'
   ...
```

3.15.2 ereg_replace

Einen Schritt weiter geht die Funktion ereg_replace() (eregi_replace() ohne Berücksichtigung von Groß- und Kleinschreibung). Hier wird der gefundene Ausdruck durch eine andere Zeichenkette ersetzt. Die Rückgabe der Funktion ist der neue String.

```
<?php
  $string = 'Hallo';
  echo ereg_replace('lo','lllo',$string);
        //gibt 'Halllo' aus
  $new = ereg_replace('^','<br />'$string);
        // setzt einen Zeilenumbruch vor den Anfang
        // des Strings
  $new = ereg_replace('\n','<br />'$string);
        // ersetzt jedes \n in $string durch  <br />
```

3.16 Benutzerdefinierte Funktionen

Der Aufbau einer Funktionsdeklaration ist denkbar simpel:

```
function funktionsname(parameter1,parameter2,...)
{
  ...Anweisungen...
}
```

3.16.1 Funktionsparameter

Wie in der Mathematik macht auch in PHP eine Funktion ohne Parameter nur in seltenen Fällen Sinn. Dabei gibt es aber mehrere Dinge zu beachten.

Parameterübergabe

In vielen Fällen ist es notwendig, an eine Funktion Werte zu übergeben. Das geschieht mittels der Parameterliste im Funktionskopf:

```
function add($param1,$param2)
{
  echo $param1 . '+' . $param2 . '=';
  return $param1+$param2;
}

echo add(12,3);

// Ausgabe : 12+3=15
```

Den so angegebenen Parametern wird bei einem Aufruf der Funktion der entsprechende Wert zugewiesen.

3.16.2 Geltungsbereich von Variablen

Eine Variable, die innerhalb einer Funktion steht, ist nur innerhalb dieser Funktion gültig. Das heißt, eine Variable gleichen Namens im Hauptprogramm hat mit der Variablen in der Funktion nichts zu tun.

```
<?php
$summe=50;

function summe($a,$b)
{
   $summe=$a+$b;
   print "Die Summe aus der Subroutine ist $summe <br>";
}

summe(50,70);
print "Die Summe im Hauptprogramm ist $summe <br>";

?>

// Ausgabe:  Die Summe aus der Subroutine ist 120
//           Die Summe aus dem Hauptprogramm ist 50
```

Globale Variablen

Wenn man möchte, dass eine Variable einer Funktion im ganzen Bereich des Programms gültig ist, dann muss man diese Variable innerhalb der Funktion mit `global` exportieren. Das sieht dann so aus:

```
<?
$summe=50;
function summe($a,$b)
{
   global $summe;
   $summe=$a+$b;
   print "Die Summe aus der Subroutine ist $summe <br>";
}
summe(50,70);
print "Die Summe im Hauptprogramm ist $summe <br>";
?>

// Ausgabe:  Die Summe aus der Subroutine ist 120
//           Die Summe aus dem Hauptprogramm ist 120
```

statische Variablen

Variablen innerhalb einer Funktion können auch statisch deklariert werden. Eine statische Variable steht nur im Geltungsbereich der Funktion zur Verfügung, aber ihr Wert geht zwischen zwei Funktionsaufrufen nicht verloren.

```
<?php

function count()
{
   static $sum = 0;

   $sum ++;

   return $sum;

echo count();
// Ausgabe : 0

echo count();
// Ausgabe : 1

?>
```

3.17 Datum und Uhrzeit

In PHP sind einige Funktionen zu Datum und Zeit integriert. Im folgenden werden die wichtigsten Funktionen vorgestellt:

```
time()          - gibt die aktuelle Zeit als sog. Unix
                  Zeitstempel zurück
mktime()        - gibt einen Unix-Zeitstempel zu einem
                  gegebenen Datum zurück
date()          - formatiert ein angegebenes Datum
checkdate()     - überprüft ein angegebenes Datum auf
                  dessen Gültigkeit
```

$date_aktuell = time();

Wir bekommen mit time() den aktuellen UNIX-Zeitstempel zurück. Das ist die Anzahl der Sekunden seit dem 1.1.1970 - 0 Uhr, dem Beginn der UNIX-Epoche. Im jetzigen Moment sind es z.B. 1108545784 Sekunden.

```
echo date("d.m.Y, H:i:s",time());
echo date("d.m.Y, H:i:s",1108545784);
```

date() erfordert verschiedene Formatierungsstrings, z.B.:

```
- d für Tage (2-stellig)
- m für Monate (2-stellig)
- Y für Jahre (4-stellig)
- H für die Stunden (2-stellig)
- i für Minuten (2-stellig)
- s für Sekunden (2-stellig)
```

Mithilfe der Funktion mktime() lässt sich jedes beliebige Datum in einen UNIX-Zeitstempel umwandeln. Die Anordnung der Argumente sollte unbedingt beachtet werden, da sonst Fehler vorprogrammiert sind:

```
int mktime ( [int Stunde [, int Minute [, int Sekunde [, int
Monat [, int Tag [, int Jahr [, int is_dst]]]]]]])
```

Beispiel: Setzen wir den Zeitstempel auf Neujahr 2007:

```
$timestamp = mktime(0, 0, 0, 1, 1, 2007);

echo date("d.m.Y, H:i:s",$timestamp);

// Ausgabe : 01.01.2007, 00:00:00
```

3.18 Zugriff auf Datenbanken

Eines der Haupteinsatzgebiete von PHP ist der Online-Zugriff auf Datenbanken.

Was ist eine Datenbank?

Vereinfacht gesagt: Eine Datenbank enthält Datensätze, die mit einer speziellen Sprache bearbeitet werden können. Zum Beispiel können bestimmte Datensätze ausgelesen werden, neue Datensätze hinzugefügt werden, Datensätze können aktualisiert oder gelöscht werden. All diese Vorgänge nennt man Abfragen.

Weit verbreitet sind sogenannte SQl-Datenbanken, d.h. Datenbanken, die mit der Sprache SQL (Abkürzung für Structured Query Language) bearbeitet werden können. PHP kann mit Datenbanken verschiedener Hersteller umgehen, besonders beliebt ist die Datenbank MySQL.

Vorteile von MySQL:

relativ schnell und kostenlos.

Um eine SQL-Abfrage mit PHP auszuführen, muß zuerst die Datenbank geöffnet werden (vergleichbar mit dem Zugriff auf eine Datei), dann wird die SQL-Befehlszeile an die Datenbank geschickt, die Antwort der Datenbank wird aufgenommen und schließlich wird die Datenbank-Verbindung wieder geschlossen.

3.18.1 SQL-Grundlagen

SQL-Datenbanken bestehen aus einer oder mehreren Tabellen. Jeder Datensatz der Datenbank ist genau eine Zeile in einer Tabelle.

Tabelle : angestellte

id	name	vorname	gebdatum	job
1	Müller	Eric	11.12.1988	1
2	Meier	Lisa	06.07.1965	2
3	Bungarz	Rolf	01.03.1976	1
4	Unger	Jasmin	05.11.1966	3

Tabelle : jobs

job	bezeichnung
1	Maurer
2	Baggerfahrer
3	Straßenfeger

Tabelle: auftraege

auftr_id	id	ort	datum
1	1	Köln	05.05.2006
2	1	Köln	06.05.2006
3	2	Köln	05.05.2006
4	2	Köln	07.05.2006
5	2	Köln	08.05.2006
6	3	Bonn	05.05.2006
7	3	Bonn	06.05.2006
8	3	Bonn	07.05.2006
...	...	...	...

3.18.2 Gebräuchliche Datentypen

Es gibt viele mögliche Datentypen in MySQL. Einige der gebräuchlichsten zeigt die folgende Tabelle:

Datentyp	Bemerkungen
`int(length)`	Integer mit einer Höchstlänge *length*
`decimal (width[,decimal_places])`	Zahl mit einer Breite *width* und einer optionalen Anzahl von Nachkommastellen (*decimal_places*)
`datetime`	Speichert ein Datum und eine Uhrzeit im Format JJJJ-MM-TT HH:MM:SS
`time`	Speichert eine Uhrzeit im Format HH:MM:SS
`date`	Speichert ein Datum im Format JJJJ-MM-TT
`timestamp`	Speichert ein Datum und eine Uhrzeit im Format JJJJMMTTHHMMSS
`varchar(length)`	Ein Textstring mit variabler Länge und der maximalen Länge *length*
`char(length)`	Ein Textstring mit fester Länge *length*
`blob`	Ein Attribut mit maximaler Datenmenge bis zu 64kByte

3.18.3 Schlüssel

Ein Primärschlüssel besteht aus einem oder mehreren Attributen, die eine Tabellenzeile eindeutig identifizieren. Primärschlüssel sind für die Aufrechterhaltung der Tabellenbezeichnungen in einer Datenbank absolut zentral. Jede Tabelle sollte einen Primärschlüssel haben. In der Tabelle *angestellte* hat die id diese Funktion. Sie identifiziert eindeutig jeden Angestellten der Tabelle. Genauso könnte jedoch auch den Name eines Angestellten ein Primärattribut sein.

3.18.4 Standardwerte

Wenn sie keinen Wert für ein Attribut angeben wird immer der DEFAULT-Wert gesetzt, wenn dieser im vorhinein definiert wurde. Bei der Tabellendefinition kann ein solcher DEFAULT-Wert angegeben werden. Wenn ein Attribut als NOT NULL definiert wurde, aber andererseits auch kein DEFAULT-Wert gesetzt wurde, hängt es von seinem Typ ab welcher Wert gesetzt wird. Bei Integer-Variablen wird automatisch der Wert 0 gesetzt, Strings werden auf den leeren String gesetzt. Damit man sich nicht bei jedem INSERT Gedanken darum machen muss welche Werte man angeben muss, sollte man regen Gebrauch von DEFAULT-Werten machen.

3.18.5 Automatisches Inkrementieren

MySQL stellt den Modifier *auto_increment* zur Verfügung, der nicht zum Standard-SQL gehört, jedoch die Verwaltung von Primärschlüsseln vereinfacht. Auch die meisten anderen Datenbankserver haben ein solches nicht-standardisiertes Feature. *auto_increment* gewährleistet, das jede Zeile einer Tabelle einen eindeutigen Primärschlüssel hat, indem es den bisherigen Höchstwert um eins erhöht als neuen Wert für die einzufügende Zeile speichert. Ist beispielsweise der höchste eingetragene Wert 10 würde der nächste Eintrag den Wert 11 bekommen. Beachten Sie dass immer nur ein Attribut einer Tabelle den Modifier *auto_increment* bekommen kann.

3.18.6 Auslesen von Datensätzen

Auslesen aller Angestellten:

```
SELECT * FROM angestellte;

// Ausgabe :
1,Müller,Eric,11.12.1988,1
2,Meier,Lisa,06.07.1965,2
3,Bungarz,Rolf,01.03.1976,1
4,Unger,Jasmin,05.11.1966,3
```

Möchte man nur gewisse Felder Auslesen kann man dies folgendermaßen

```
SELECT name,vorname FROM angestellte;

// Ausgabe :
Müller,Eric
Meier,Lisa
Bungarz,Rolf
Unger,Jasmin
```

Es kann auch eine bedingte Abfrage an die Datenbank gestellt werden, z.B. möchte man alle Angestellten auslesen die den Job mit der id 1 haben

```
SELECT * FROM angestellte WHERE job=1;

// Ausgabe :
1,Müller,Eric,11.12.1988,1
3,Bungarz,Rolf,01.03.1976,1
```

ORDER BY

Möchte man diese Datensätze nach dem Feld "name" sortiert ausgeben, kann man dies mit der einfachen Angabe

```
SELECT * FROM angestellte WHERE job=1 ORDER BY name;

// Ausgabe :
3,Bungarz,Rolf,01.03.1976,1
1,Müller,Eric,11.12.1988,1


// Entgegengesetzte Sortierung
SELECT * FROM angestellte WHERE job=1 ORDER BY name DESC;

// Ausgabe :
1,Müller,Eric,11.12.1988,1
3,Bungarz,Rolf,01.03.1976,1
```

GROUP BY

Die GROUP BY – Klausel unterscheidet sich insofern von ORDER BY, als sie nicht Daten für die Ausgabe sondern viel früher im Abfrageprozess sortiert bzw. gruppiert. Der Zweck ist, eine Zusammenfassung gleicher Einträge zu erreichen. Durch diese Zusammenfassung lassen sich am einfachsten Eigenschaften wie Maximal – oder Minimalwerte, Durchschnitte und Summen ermitteln.

```
SELECT ort, COUNT(*) FROM auftraege GROUP BY ort;
```

Diese Abfrage hat zu Folge, dass zunächst die Zeilen der Tabelle auftraege nach ort sortiert wird und dann alle Zeilen mit übereinstimmenden ort-Werten zusammengefasst werden. Die Ausgabe beinhaltet den Stadtnamen zusammen mit der mittels COUNT(*) ermittelten Anzahl der Aufträge in dieser Stadt.

city	COUNT(*)
Bonn	3
Köln	5

Die GROUP BY – Klausel kann verschiedene Eigenschaften der Zusammengefassten Zeilen herausfinden. Im folgenden sind die fünf gängigsten Funktionen aufgeführt. Eine Abfrage kann mehrere dieser Funktionen beinhalten.

Funktion	Beschreibung
`AVG()`	Findet den Durchschnittswert eines nummerischen Attributs in einer Menge
`MIN()`	Findet den kleinsten Wert eines String-Attributs oder eines nummerischen Attributs in einer Menge
`MAX()`	Findet den größten Wert eines String-Attributs oder eines nummerischen Attributs in einer Menge
`SUM()`	Findet die Gesamtsumme eines nummerischen Attributs
`COUNT()`	Zählt die Zeilen in einer Menge

HAVING

Die HAVING – Klausel ermöglicht es, Daten anhand von Bedingungen zu Gruppieren. Folgendes Beispiel veranschaulicht die Vorgehensweise von HAVING:

```
SELECT ort, COUNT(*), MIN(datum) FROM auftraege GROUP BY ort
                    HAVING count(*) > 4;
```

Als Antwort auf diese Abfrage bekommt man den Stadtnamen zusammen mit der Anzahl an Aufträgen und dem Frühesten Datum der Aufträge der Städte mit mehr als 4 Aufträgen.

city	COUNT(*)	MIN(datum)
Köln	5	05.05.2006

DISTINCT

Angenommen man möchte herausfinden, in welchen Städten Aufträge angenommen wurden. Mit der folgenden Abfrage bekommen wir alle Städte, in denen Aufträge bearbeitet wurden:

```
SELECT ort FROM auftraege;
```

Das Problem ist nur: Wir bekommen jede Stadt genau so oft ausgegeben wie Aufträge vergeben wurden. Wurden beispielsweise in Köln 223 Aufträge bearbeitet erhalten wir mit dieser Abfrage genau 223 mal Köln als Ausgabe. Da dies nicht Sinnvoll ist möchten wir erreichen das identische Zeilen nur einmal ausgegeben werden.

Mit folgender Zeile erreichen wir genau dieses:

```
SELECT DISTINCT ort FROM auftraege;
```

Diese Abfrage hätte genau die selbe Wirkung wie:

```
SELECT ort FROM auftraege GROUP BY ort;
```

Ausgaben in MySQL begrenzen

Der Operator LIMIT ist MySQL-spezifisch und ist dazu da, die Größe der Ausgabe zu steuern. Die folgende Abfrage liefert beispielsweise nur die ersten fünf Zeilen der Tabelle auftraege:

```
SELECT * FROM auftraege LIMIT 5;
```

Dieser Ausdruck kann noch verfeinert werden indem man die Startzeile angibt:

```
SELECT * FROM auftraege LIMIT 100,5;
```

Diese Abfrage hätte zur Folge, dass die Zeilen 100 bis 104 der Tabelle ausgegeben werden.

Besonders praktisch ist dieses Vorgehen in der Webapplikations- Entwicklung wenn große Datenbestände sortiert ausgegeben werden sollen und dies Seitenweise beispielsweise mit maximal 30 Einträgen pro Seite angezeigt werden sollen.

Wenn Sie ab einer bestimmten Zeile alle restlichen Zeilen anzeigen möchten können Sie dies veranlassen indem Sie den zweiten Parameter auf -1 setzen.

```
SELECT * FROM auftraege LIMIT 100,-1;
```

Der Operator LIMIT wird immer an das Ende einer SQL-Anweisung angehängt, also hinter die optionalen WHERE-, GROUP BY-, HAVING- und ORDER BY-Klauseln.

3.18.7 JOIN-Abfragen

Oft möchte man Daten abfragen, die auf Beziehungen zwischen zwei oder mehr Tabellen beruhen. Möchte man beispielsweise wissen welcher Mitarbeiter einen bestimmten Auftrag erledigt hat oder welcher Mitarbeiter wie viele Aufträge in einer bestimmten Stadt bearbeitet hat, so sind JOIN-Abfragen das Mittel der Wahl. In SQL gleicht eine JOIN-Abfrage Zeilen aus zwei oder mehr Tabellen anhand einer Bedingung ab, die in einer WHERE-Klausel formuliert worden ist und gibt nur Zeilen aus die dieser Bedingung entsprechen.

Elementare JOINs

Nehmen wir als Beispiel folgende Tabellen. Abfragen wollen wir den Namen, Vornamen und die Bezeichnung des Jobs aller Angestellten in einer einzigen Abfrage.

Tabelle : angestellte

id	name	vorname	gebdatum	job
1	Müller	Eric	11.12.1988	1
2	Meier	Lisa	06.07.1965	2
3	Bungarz	Rolf	01.03.1976	1
4	Unger	Jasmin	05.11.1966	3

Tabelle : jobs

job	bezeichnung
1	Maurer
2	Baggerfahrer
3	Straßenfeger

Das Ergebnis dieser Abfrage sollte ungefähr so aussehen:

```
Müller Eric Maurer
Meier Lisa Baggerfahrer
Bungarz Rolf Maurer
Unger Jasmin Straßenfeger
```

Mit einer JOIN-Abfrage ist genau dies möglich. Folgende JOIN-Abfrage realisiert genau die oben geforderte Abfrage:

```
SELECT name, vorname, bezeichnung FROM angestellte, jobs
         WHERE angestellte.job = jobs.job;
```

Natürlich ist es auch hier möglich die bereits bekannten Angaben zum Sortieren der Ausgabe zu machen. Folgende Erweiterung führt zu einer nach dem Namen des Angestellten sortierten Liste:

```
SELECT name, vorname, bezeichnung FROM angestellte, jobs
         WHERE angestellte.job = jobs.job
         ORDER BY name;

// Ausgabe:
Bungarz Rolf Maurer
Meier Lisa Baggerfahrer
Müller Eric Maurer
Unger Jasmin Straßenfeger
```

Es ist möglich und in einigen Fällen wichtig die Tabellenbezeichner vor den abzufragenden Attributen mit anzugeben. Wichtig ist dies, wenn Attribute in den abgefragten Tabellen den gleichen Namen haben.

```
SELECT angestellte.name, angestellte.vorname, jobs.bezeichnung
         FROM angestellte, jobs
         WHERE angestellte.job = jobs.job;
```

Generell können JOIN-Abfragen auch auf mehr als zwei Tabellen gemacht werden. Das folgende Beispiel zeigt wie dies aussehen könnte:

```
SELECT angestellte.name, auftraege.ort, auftraege.datum,
jobs.bezeichnung
         FROM angestellte, auftraege, jobs
         WHERE
         auftraege.id = angestellte.id AND
         jobs.job = angestellte.job AND
         jobs.job = 1;
```

Die Ausgabe die diese Abfrage erzeugt listet den Namen aller Maurer auf die bereits Aufträge bearbeitet haben, zusätzlich werden der Einsatzort und das Einsatzdatum ausgegeben.

3.18.8 Einfügen von Datensätzen

Neue Datensätze werden folgendermaßen hinzugefügt:

```
INSERT angestellte (name,vorname,gebdatum,job)
       VALUES ('Jakobi','Frank','15.03.1955',1);
```

id	name	vorname	gebdatum	job
1	Müller	Eric	11.12.1988	1
2	Meier	Lisa	06.07.1965	2
3	Bungarz	Rolf	01.03.1976	1
4	Unger	Jasmin	05.11.1966	3
5	Jakobi	Frank	15.03.1955	1

3.18.9 Ändern von Datensätzen

Man kann bereits in der Datenbank vorhandene Einträge folgendermaßen ändern:

```
UPDATE angestellte SET job=2 WHERE id=1;
```

id	name	vorname	gebdatum	job
1	Müller	Eric	11.12.1988	2
2	Meier	Lisa	06.07.1965	2
3	Bungarz	Rolf	01.03.1976	1
4	Unger	Jasmin	05.11.1966	3
5	Jakobi	Frank	15.03.1955	1

3.18.10 Löschen von Datensätzen

Das Löschen von Datensätzen erfolgt mit der Anweisung DELETE:

```
DELETE FROM angestellte WHERE id=5;
```

id	name	vorname	gebdatum	job
1	Müller	Eric	11.12.1988	2
2	Meier	Lisa	06.07.1965	2
3	Bungarz	Rolf	01.03.1976	1
4	Unger	Jasmin	05.11.1966	3

Vorsicht! Vergisst man die WHERE-Klausel in der Anweisung, so werden alle Datensätze der Tabelle "angestellte" gelöscht!

3.19 MySQL in PHP

Um jetzt mit PHP auf die Daten in unserer Datenbank zugreifen zu können, müssen wir zunächst eine Verbindung herstellen. Dies erfolgt über die Funktion **mysql_connect()** und noch vor allen anderen Inhalten des Dokumentes.

Der Befehl benötigt den Servernamen, den Benutzernamen und das Kennwort. Der Servername ist häufig "`localhost`". Immer mehr Webhoster haben aber ihre Datenbanken auf einen eigenen Server ausgelagert. In dem Fall muss dann der Pfad zu dem Server angegeben werden.

Verbindung zum Datenbankserver herstellen:

```
<?
  mysql_connect("loclahost","user","password");
?>
```

Als nächstes benötigen wir eine Verbindung zu der Datenbank, auf der wir arbeiten wollen. Dafür gibt es die Funktion **mysql_select_db()**.

```
<?
  mysql_select_db("meinedb");
?>
```

Da diese Daten in den meisten Fällen gleich bleiben (außer man verwendet mehrere Datenbanken bzw. Server), ist es sinnvoll, diesen Programmteil in einer speziellen PHP-Datei zu speichern und diese anschließend über die Funktion include() oder require() einzubinden.

Sollte irgend etwas bei der Verbindung zur Datenbank nicht funktionieren, ist es sinnvoll, sich Warnmeldungen anzeigen zu lassen, damit man weiß, wo der Fehler auftrat. Diese erfolgen jeweils vor dem Semikolon und sehen dann so aus:

```
<?php
  mysql_connect("localhost","user","password") or die
              ("Keine Verbindung moeglich");

  mysql_select_db("meinedb") or die
                ("Die Datenbank existiert nicht");
?>
```

mysql_query()

Die Funktion mysql_query() sendet eine Anfrage zur derzeit aktivierten Datenbank. Übergeben werden muss dieser Funktion ein String, welcher die SQL-Abfrage enthält. Ausschließlich für die SQL-Anweisungen `SELECT`, `EXPLAIN`, `SHOW` oder `DESCRIBE` liefert mysql_query() im Erfolgsfall eine Ressourcen-Kennung. Bei allen anderen Anweisungen wird im Erfolgsfall **`TRUE`** zurückgegeben. Ist bei der Verarbeitung der Abfrage ein syntaktischer Fehler erkannt worden wird **`FALSE`** zurückgegeben. mysql_query() übergibt mximal eine SQL-Abfrage an die Datenbank. Das Aneinanderhängen mehrerer Abfragen ist nicht möglich.

```
<?php
  $abfrage = "INSERT INTO angestellte
                        (name,vorname,gebdatum,job)
                  VALUES
                        ('Jakobi','Frank','15.03.1955',1)";

  $ergebnis = mysql_query($abfrage);

  if( !$ergebnis )
  {
     echo 'Fehler in der Syntax';
  }
?>
```

! SQL-Abfragen sollten nicht mit einem Semikolon abgeschlossen werden

mysql_fetch-array()

Die Funktion mysql_fetch_array() erwartet als Übergabeparameter ein positives Ergebnis von mysql_query(). Liefert ein assoziatives Array das dem aktuellen Datensatz entspricht oder **FALSE**, wenn keine weiteren Datensätze vorliegen. Die Elemente des Arrays haben den Namen der jeweiligen Feldnamen der Datenbanktabelle.

```
<?php
  $abfrage = "SELECT name,nachname FROM angestellte";
  $ergebnis = mysql_query($abfrage);

  while($row = mysql_fetch_array($ergebnis))
  {
    echo $row['name'] . ',' . $row['vorname'];
  }
?>

// Ausgabe :
Müller,Eric
Meier,Lisa
Bungarz,Rolf
Unger,Jasmin
```

! Feldnamen sind case sensitiv (Name!=name)

mysql_num_rows()

Die Rückgabe dieser Funktion ist die Anzahl der Datensätze einer Ergebnismenge. Diese Funktion ist nur für SELECT-Abfragen gültig.

```
<?php
  $abfrage = "SELECT name,nachname FROM angestellte";
  $ergebnis = mysql_query($abfrage);

  echo mysql_num_rows($ergebnis);

?>

// Ausgabe : 4
```

mysql_affected_rows()

Die Rückgabe dieser Funktion ist die Anzahl der von der SQL-Abfrage betroffenen Datensätze. Diese Funktion ist nur für INSERT, UPDATE und DELETE-Abfragen gültig.

```
<?php
  $abfrage = "DELETE FROM angestellte WHERE id > 2";
  $ergebnis = mysql_query($abfrage);

  echo mysql_affected_rows($ergebnis);

?>

// Ausgabe : 3
```

mysql_error()

mysql_error() liefert eine detaillierte Fehlermeldung im Falle eines Fehlers. Ansonsten wird ein leerer String ' ' zurückgeliefert. Es wird immer die Fehlermeldung der letzten MySQL-Abfrage ausgegeben. (siehe auch mysql_errno())

```
<?php
  $abfrage = "DELETE FROM angeslte WHERE id > 2";
  $ergebnis = mysql_query($abfrage);

  echo mysql_error();

  ...

?>

// Ausgabe : Table 'angeslte' doesn't exist
```

3.20 Gefahr durch MySQL-Injections

Eine große Gefahr bei der Entwicklung von Webanwendungen ist die der SQL-Injections. Hierunter versteht man schadhaftes einschleusen von SQL-Abfragen, oder Teilen davon. Diese SQL-Abfragen erlauben einem Angreifer vielfältige Möglichkeiten.

Schauen Sie sich folgendes Beispiel an. Ein Benutzerlogin wird bei dieser Anwendung mittels Benutzernamen und Passwort realisiert. Hierbei werden die Felder `username` und `password` innerhalb eines Formulars eingegeben und per POST an die folgende PHP-Datei gesendet. Nun wird eine SQL-Abfrage an die Benutzerdatenbank abgesetzt und im Falle eines Ergebnisses der Login gewährt.

```
<?php
   ...

  $abfrage = 'SELECT * FROM user
                    WHERE
                          username="'.$_POST['username'].'"
                    AND   password="'.$_POST['password'].'"';

  $ergebnis = mysql_query($abfrage);

   if( mysql_num_rows($ergebnis) > 0 )
   {
         echo 'Login war erfolgreich';
         ...

?>
```

Was würde nun passieren wenn die Variable $_POST['password'] den folgenden Inhalt hätte: `'irgendwas" OR "1"="1'`?

Hieraus würde die Folgende SQL-Abfrage resultieren:

```
SELECT * FROM user
         WHERE username="user"
         AND password="irgendwas"
         OR "1"="1";
```

Diese SQL-Abfrage liefert immer ein positives Ergebnis! Der Angreifer hätte den Loginmechanismus ausgehebelt.

3.20.1 Schutz vor SQL-Injections

Die wichtigsten Regeln für jeden Webentwickler sind:

- Alle Eingaben, welche nicht selbst erzeugt wurden ($_GET, $_POST, etc.) sind generell **nicht vertrauenswürdig und gelten als potenziell verseucht**!
- Alle Eingaben müssen **vor der Weiterverarbeitung** eingehend geprüft werden.
- Sofern möglich alles Verbieten, was nicht explizit erlaubt ist (z.B. Texteingaben mit regulären Ausdrücken filtern, [A-Za-z0-9:space:]).
- Verwenden gängiger Funktionen für das Maskieren spezieller Zeichen.

mysql_real_escape_string()

Die wichtigste Funktion für den Schutz vor SQL-Injections ist mysql_real_escape_string(). Diese Funktion maskiert spezielle Zeichen (\x00, \n, \r, \, ', " und \x1a) innerhalb eines ihr übergebenen Strings für die Verwendung in einer SQL-Anweisung und gibt den ungefährlichen String zurück.

Diese Funktion muss immer verwendet werden, wenn Daten in einer SQL-Anweisung übermittelt werden sollen!

```
<?php
   $username = mysql_real_escape_string($_POST['username']);
   $password = mysql_real_escape_string($_POST['password']);

   $abfrage = 'SELECT * FROM user
               WHERE
                     username="'.$username.'"
               AND   password="'.$password.'"';

  $ergebnis = mysql_query($abfrage);

   ...
```

Was würde nun passieren wenn die Variable $_POST['password'] den folgenden Inhalt hätte: `'irgendwas" OR "1"="1'`?

Hieraus würde die Folgende SQL-Abfrage resultieren:

```
SELECT * FROM user WHERE
               username="sas"
         AND
               password="irgendwas\" OR \"1\"=\"1"
```

Alle schädlichen Zeichen sind mit escape-Strings entschärft worden. Dieser Angriffsversuch wäre gescheitert.

3.21 Cookies

Als Cookies werden kurze Informationen bezeichnet, die der WWW-Server beim Client (Betrachter) hinterlassen kann. Sie sind vor allem beim elektronischen Einkaufen im WWW von Bedeutung, genauer gesagt sind sie immer dann von Bedeutung, wenn der Server weitergehende Information vom Betrachter haben muß, als nur die Internet-Adresse, die eine Seite anfordert.

Cookies bestehen aus folgender Information:

Information/Parameter	Bemerkungen
`name`	Name des Cookies (wird für Zugriffe auf das Cookie benötigt)
`value`	Wert (nutzbarer Inhalt, Variable)
`expires`	Verfallsdatum
`path`	Nur über Seiten, die in diesem Verzeichnis, oder Unterverzeichnissen davon liegen, darf das Cookie angefordert werden ('/' für gesamte Domain)
`domain`	Adressraum der Server, die das Cookie wieder lesen dürfen
`secure`	Wenn gesetzt, darf das Cookie nur über verschlüsselte Informationskanäle übertragen werden (HTTPS)
`httponly`	Wenn auf true gesetzt, so ist das Cookie ausschließlich via http aufrufbar. (JavaSkript, etc. sind damit ausgeschlossen)

setcookie()

Die Funktion setcookie() definiert ein mit dem HTTP-Header zu übertragendes Cookie. Dies bedeutet unter anderem, dass diese Funktion vor allen Ausgaben des Skriptes aufgerufen werden muss. Die in obiger Tabelle genannten Parameter können in genau dieser Reihenfolge an setcookie() übergeben werden. Alle Übergabeparameter, mit Ausnahme von `name`, der Funktion setcookie() sind optional.

Speichern Sie niemals sensible Daten innerhalb eines Cookies, da diese auf dem Client gespeichert werden und somit nicht vor Zugriffen geschützt sind.

Beispiel: Wir wollen feststellen, ob ein Betrachter eine Seite schon einmal aufgerufen hat. Dazu setzen wir beim ersten Aufruf der Seite ein Cookie.

```
<?php
        $t = time()+3600*24*10;
        setcookie("mycookie","Schon besucht",$t,"/test",".de");
?>
<HTML>
...
```

Dieser Befehl setzt ein Cookie mit dem Namen mycookie auf den Wert "Schon besucht" mit einem Verfallsdatum von 10 Tagen. Lesen darf dieses Cookie jeder Server mit der Endung .de, allerdings nur aus Dateien, die in einem Verzeichnis /test, oder Unterverzeichnissen davon liegen.

Das Verfallsdatum des Cookie muß in Sekunden seit dem 1. Januar 1970 angegeben werden. Die Funktion time() gibt die Sekunden seit dem 1. Januar 1970 der

momentanen Zeit an. Da die Berechnung der Sekunden nicht immer ganz übersichtlich ist, empfiehlt sich die Verwendung einer Funktion:

```
function tage($anzahl) {
       $sekunden = time()+ 3600 * 24 * $anzahl;
       return $sekunden;
}

setcookie("mycookie","Schon besucht",tage(10),"/test",".de");
```

Cookieinhalt auslesen

Cookies lesen ist mit PHP wieder sehr einfach realisiert. Der Inhalt des Cookies ist in einer assoziativen Array-Variablen mit dem Namen des Cookies enthalten (vgl. $_GET, $_POST, etc.):

```
echo $_COOKIE['mycookie'];

// Ausgabe: Schon besucht
```

Cookie löschen

Um ein Cookie wieder zu löschen kann man einfach das Verfallsdatum des Cookies auf einen Wert setzen, welcher in der Vergangenheit liegt.

```
// Setzen der Verfallszeit auf einen Wert in der Vergangenheit:
setcookie("mycookie","",time()-100,"/test",".de");
```

3.22 Sessions

Ein grundlegendes Kennzeichen des Internets ist die zustandslose Interaktion zwischen dem Browser und dem Webserver. HTTP ist ein zustandsloses Protokoll. Alle Requests die an den Webserver gesendet werden sind voneinander Unabhängig. Diese zustandslose Natur ist für Anwendungen geeignet, die es Benutzern ermöglichen Dokumentsammlungen in beliebiger Reihenfolge zu durchsuchen. Anwendungen, die eine komplexere Interaktion mit Benutzern ermöglichen sind hiermit nicht möglich. Zustandsbehaftete Webdatenbank-Applikationen können mit Sessions realisiert werden. Hierzu wird auf dem Clientrechner ein Session-Cookie gespeichert. Der Wert des Cookies ist die Session ID (Zufallswert 2^32) dieser Session. Diese ID identifiziert diesen Browser (Benutzer) eindeutig. Auf dem Server liegt für jede Session ID eine Datei mit allen dieser Session zugeordneten Variablen und Eigenschaften. Sensible Daten sind auf diese Weise recht gut vor nicht erlaubten Veränderungen geschützt. Über den unsicheren Kanal wird ausschließlich die Session ID gesendet.

3.22.1 Eine Session starten

Vor jeder Datei mit der Endung ".php" in der die Sessionvariablen verwendet werden sollen, ist es nötig die Session zu starten. Dies muss vor dem HTML-Tag "<head>" passieren. Sollte Ein User im selben Browser auf einer anderen Seite surfen und dann wieder zurückkommen, so ist die Session noch vorhanden. Die Session geht in der Regel verloren, wenn der Browser komplett geschlossen wird. Zum Erzeugen einer Session wird die Funktion session_start() verwendet. Jede Session identifiziert eindeutig die Interaktion eines bestimmten Webbrowsers mit einer Webdatenbank-Anwendung.

```
<?php
   //starten der Session
   session_start();
?>
```

Wird die Funktion session_start() das erste mal ausgeführt, so wird automatisch eine neue Session-ID erzeugt und eine leere Datei in der die Session-Variablen gespeichert werden auf dem Server abgelegt. Des Weiteren wird dem Client ein Cookie geschickt, welches die Session-ID enthält. Der von PHP generierte Session-Identifier ist ein Zufallsstring, der aus 32 hexadezimalen Ziffern beseht.

In jedem Dokument, welches ein Teil der Session sein soll (welches also auf Sessionvariablen zugreifen können soll) muss session_start() aufgerufen werden!

3.22.2 Session-Variablen verwenden

Die Funktion session_start() wird ebenfalls verwendet um eine bereits vorhandene Session zu finden. Beim Ausführen von session_start() wird zunächst im Browser nach dem entsprechenden Cookie gesucht. Wird ein reguläres, gültiges gefunden kann die dazugehörige Session verwendet werden, ist keine gültige Session vorhanden, wird eine neue eröffnet. Nachdem session_start() ausgeführt wurde führt PHP automatisch das global verfügbare, assoziative Array $_SESSION. Mittels dieses Arrays kann nun auf die Session-Variablen zugegriffen werden. Im Folgenden soll ein Beispiel das Verwenden von Session-Variablen verdeutlichen:

```
<?php
   //starten der Session
   session_start();

   //belegen einer Variablen mit dem aktuellen Zeitstempel
   $last_visit = $_SESSION['time'];
   $_SESSION['time'] = time();

   echo 'Letzter Besuch vor ' . $_SESSION['time']-$last_visit .
'Sekunden';
?>
```

3.22.3 Session ID auslesen

Die Funktion session_id() gibt im dem Fall, wenn eine Session besteht die Session ID zurück. Falls keine Session existiert wird der leere String '' zurückgegeben.

3.22.4 Session ID ändern

Mithilfe der Funktion session_regenerate_id() lässt sich eine bereits gesetzte Session ID neu erstellen. Alle bisher gesetzten Sessioninformationen werden übernommen.

```
<?php
   session_start();
   ...
   // Derzeitige Session ID: 751d031f7330acff519faf066aef8715

   session_regenerate_id();

   echo session_id();

   // Ausgabe: bff103f6726c71bc1bd1d8eb51dadb31
?>
```

3.22.5 Session-Variablen entfernen

Um eine Session-Variable zu entfernen verwendet man die Funktion unset(). Um alle Session-Variablen zu entfernen kann man diese Funktion auf das gesamte Array anwenden. Dies hätte zu Folge, dass zwar alle Variablen, die zu einer Session gehören gelöscht werden, die Session selbst bliebe jedoch bestehen.

```
<?php
   // löschen der Session-Variable time
   unset( $_SESSION['time'] );

   // loschen des gesammten Arrays
   unset( $_SESSION );

   // oder mittels Überschreiben
   $_SESSION = array();
?>
```

3.22.6 Eine Session beenden

An irgendeinem Punkt einer Anwendung sollten Sessions zerstört werden. Beispielsweise sollte ein Aufruf der Funktion session_destroy() erfolgen, wenn ein Benutzer sich aus einer Anwendung ausloggt. Wird die Funktion session_destroy() ausgeführt werden die von der zugehörigen Session erzeugten Variablen und Session-Dateien auf dem Server gelöscht. Erhalten bleibt jedoch das im Browser gespeicherte Session-Cookie.

```
<?php
   // eine Session MUSS bestehen bevor session_destroy()
   // aufgerufen wird
   session_start();

   // Beenden des Session
   session_destroy();

   // Umleiten auf eine Logout-Page zur Bestätigung
   header('Location: logout.html');
?>
```

3.22.7 Erstellen Sessionbasierter Anwendungen

Ein Beispiel:

Zunächst erstellen wir ein Formular für die Eingabe des Benutzernamens und des Passwortes. Bei klick auf den Button mit der Aufschrift „Anmelden“ werden die Eingaben per POST an das Dokument „validate.php“ gesendet.

```
login.html:

<html>
<head>
<title>Login</title>
</head>

<body>
<form method="post" action="validate.php">
Benutzername<br>
<input type="text" size="10" name="username">
<br><br>
Passwort<br>
<input type="password" size="10" name="password">

<br><br>
<input type="submit" value="Anmelden">
</form>
</body>
</html>
```

Die übergebenen Variablen werden zunächst recht einfach Überprüft. Hier soll es zunächst ausreichen auf leere Eingaben zu Überprüfen. Danach wird automatisch mittels einer Datenbankabfrage Überprüft ob der Benutzer zugriffsberechtigt ist. Wenn ja wird sein Benutzername und seine IP in der Session-Variablen gespeichert.

validate.php:

```
<?php
   session_start();
?>
<?php
   include("sql_connect.php");
   $username = $_POST['username'];
   $password = $_POST['password'];

   if(!isset($username) || !isset($password) )
   {
         header("Location: login.php");
         exit;
   }
   else
   {
         $query = 'SELECT username,password FROM user
                   WHERE username="' . $username . '"';

         $result = mysql_query($query);
         while( $row = mysql_fetch_array($result) )
         {
               if( $row["username"] == $username
                     && $row["password"] == $password )
               {
                     $_SESSION["username"] = $username;
                     $_SESSION["remoteip"] =
                                    $_SERVER["REMOTE_ADDR"];
                     header("Location: welcome.php");
                     exit;
               }
               else
               {
                     header("Location: login.php");
                     exit;
               }
         }
         header("Location: login.php");
         exit;
   }
?>
```

Ein Willkommensbildschirm zeigt dem ordnungsgemäß eingelogten Benutzer an das er nun eingelogt ist. Es wird bei jedem Aufruf von welcome.php geprüft ob der Benutzer noch eingelogt ist oder ob ein nicht authentisierter Benutzer sich Zugang erschleichen will. Ist dies der Fall wir automatisch zu login.php umgeleitet.

```
welcome.php:

<?php
   session_start();
?>
<?php
   include("sql_connect.php");
   if( !isset($_SESSION["username"] ) )
   {
         header("Location: login.php");
         exit;
   }
   else
   {
         echo "Willkommen " . $_SESSION["username"];
         echo "<br><br>Deine IP ist die : " .
                                 $_SESSION["remoteip"];

         echo "<br><br>Zum ausloggen klicken Sie bitte ";
         echo "<a href='logout.php'>hier</a>";
   }
?>
```

Nach klick auf „hier“ wird das Skript logout.php ausgeführt. Dies veranlasst das Beenden der Session und eine Umleitung zum Skript login.php.

```
logout.php:

<?php
   session_start();
   session_destroy();
   header("Location: login.php");
?>
```

3.22.8 Sessions kapern

Wie bereits erwähnt identifiziert sich eine Session ausschließlich anhand ihrer Session ID. Zwar ist es für einen Angreifer schwierig an eine gültige Session ID heranzukommen, unmöglich ist dies jedoch nicht. Wäre also ein Angreifer in der Lage an die Session ID eines Benutzers heranzukommen könnte er auf der entsprechenden Anwendung arbeiten ohne sich dort zu authentifizieren.

Es gibt jedoch einige Möglichkeiten sich bei der Entwicklung einer Anwendung vor derartigen Angriffen zu schützen. Eine sehr wirkungsvolle Methode ist das Speichern der IP-Adresse des Benutzers der Seite (Session). Falls sich die IP-Adresse des anfragenden Rechners innerhalb einer Session ändert muss diese Session sofort ungültig gemacht werden (session_destroy()). Eine auf diese Weise gesicherte Session lässt einem Angreifer kaum Spielraum, da er nicht nur die Session ID kapern muss, sondern zusätzlich auch die IP-Adresse des Benutzers. Das kapern der IP ist in den allermeisten Fällen unmöglich. Es ist denkbar den Schutz zu erhöhen indem weitere spezielle Eigenschaften des Benutzers gespeichert und bei jedem Aufruf der Applikation verglichen werden.

! Ohne weitere Sicherungsmaßnahmen sind Sessions leicht angreifbar!

3.23 HTML-Templates mit PEAR

PEAR ist eine Sammlung von PHP Klassen und Funktionen. Ist PEAR installiert, so können neue Klassen auf einfache Weise installiert werden. Eine Klasse zur Behandlung von HTML Templates stellt HTML_Template_IT dar. Sie ermöglicht es, die Trennung von Design und PHP-Code vorzunehmen, bzw. wieder zusammenzubringen.

Das folgende Beispiel zeigt den Aufbau einer einfachen Template-Datei. Zu beachten ist, das die Dateiendung von Template-Dateien nicht *.html* sondern *.tpl* ist. Template-Dateien enthalten reinen HTML-Code inklusive aller Formatierungselemente. Ein Template enthält im Unterschied zu reinen HTML-Seiten jedoch in geschweifte Klammern gefasste Platzhalter. An Stelle dieser Platzhalter werden später durch den PHP-Parser die entsprechenden Werte eingefügt.

```
// template.tpl

<html>
<head>
   <title>Mein erstes Template</title>
</head>

<body>

   <table>
   <!-- BEGIN TEST -->
   <tr>
         <td>{NAME}</td>
         <td>{VORNAME}</td>
   </tr>
   <!-- END TEST -->
   </table>

</body>
</html>
```

Ein weiterer Unterschied zu gewöhnlichen HTML-Seiten ist, dass ein Template Kommentare enthält, die die Tags BEGIN und END enthalten. Diese Kommentarblocks bilden Paare, die jeweils denselben Namen haben, mit dem ein Block definiert wird. In

unserem Beispiel haben wir einen Block mit dem Namen (Label) TEST definiert. Blöcke mit gleichem Namen dürfen mehrfach in einem Dokument definiert werden. Unterschiedliche Blöcke dürfen außerdem ineinander Verschachtelt werden.

Im folgenden wird das Skript gezeigt welches das Befüllen des Templates vornimmt. Zu beachten ist, dass dieses PHP-Skript keinerlei Ausgabefunktionen (echo, print, ...) enthält. Alle für die Ausgabe im Browser bestimmten Variablen werden zur Ausführung in das Template geparsed.

```
// template.php

<?php

   require_once("HTML/Template/IT.php");

   $template = new HTML_Template_IT("./");

   $template->loadTemplatefile("templates.tpl",true,true);

   $name = array("Müller","Schuster","Meier");
   $vorname = array("Hans","Egon","Lisa");

   $i = 0;
   while($i < 3)
   {
         $template->setCurrentBlock("TEST");
         $template->setVariable("NAME",$name[$i]);
         $template->setVariable("VORNAME",$vorname[$i]);
         $template->parseCurrentBlock();
         $i++;
   }

   $template->show();

?>


// Ausgabe:
Müller   Hans
Schuster Egon
Meier    Lisa
```

Folgende Tabelle erklärt die einzelnen Schritte und Funktionen die für das Erstellen eines zum Befüllen eines Templates gedachten PHP-Skriptes benötigt werden.

Funktion	Bemerkungen
`require_once("HTML/Template/IT.php");`	Laden der Integrated Template-Klasse von PEAR
`$template = new HTML_Template_IT("./");`	Anlegen eines neuen IT-Templates und zuweisen eines Pfades zu den Template-Files. In diesem Beispiel liegen die Templates in dem selben Ordner wie die PHP-Skripte
`$template->loadTemplatefile("templates.tpl",true,true);`	Laden des Templates welches von diesem Skript befüllt werden soll Ist der zweite Parameter auf true gesetzt werden nicht geparste Platzhalter gelöscht Ist der dritte Parameter auf true gesetzt werden nicht geparste Blöcke gelöscht
`$template->setCurrentBlock("TEST");`	Setzen des Codeblocks auf dem gearbeitet werden soll
`$template->setVariable("NAME",$name[$i]);`	Setzen einer Templatevariable auf den Wert des zweiten Parameters, dieser Wert wird im Template eingefügt
`$template->parseCurrentBlock();`	Diese Methode verarbeitet den aktuell ausgewählten Block
`$template->show();`	Veranlasst die vorher geparsten Blöcke im Browser auszugeben

3.23.1 Verschachtelte Blöcke

Wie bereits erwähnt lassen sich Template-Blöcke ineinander Verschachteln. Bei der Verwendung verschachtelter Templates muss jedoch eine einfache Regel beachtet werden: Der innerste Block muss zuerst geparst werden, dann der zweite von innen und so weiter, bis zuletzt der äußerste Block geparst wurde. Folgende Schema verdeutlicht diese Regel:

```
$template->setCurrentBlock("BLOCK1");

(Wird zuletzt geparsed)

  $template->setCurrentBlock("BLOCK2");

  (Wird als drittes geparsed)
  $template->parseCurrentBlock();

  $template->setCurrentBlock("BLOCK3");

  (Wird als zweites geparsed)

    $template->setCurrentBlock("BLOCK4");

    (Wird zuerst geparsed)

    $template->parseCurrentBlock();

  $template->parseCurrentBlock();
$template->parseCurrentBlock();
$template->show();
```

Der PHP-Code zu diesem Beispiel müsste also folgendermaßen aussehen:

```
<?php

   ...

   $template->setCurrentBlock("BLOCK4");
   // 'Füllen' der Templatevariablen deses Blocks
   // $template->setVariable(...);
   ...
   $template->parseCurrentBlock();


   $template->setCurrentBlock("BLOCK3");
   ...
   $template->parseCurrentBlock();


   $template->setCurrentBlock("BLOCK2");
   ...
   $template->parseCurrentBlock();


   $template->setCurrentBlock("BLOCK1");
   ...
   $template->parseCurrentBlock();


   $template->show();

   ...

?>
```

3.23.2 Templates mit Formularelementen

Natürlich ist es möglich beliebig komplexe Formulare mit Templates zu erstellen. Ein großer Vorteil hieran ist die einfache Handhabung und eine sehr gut les- und wartbarer Code. Erweiterungen und Anpassungen können hier leichter als in reinen PHP-Dateien vorgenommen werden. Im folgenden soll ein Beispiel das Erstellen eines Formulars verdeutlichen.

```
<html>
<head>
   <title>Mein erstes Formular</title>
</head>

<body>

   <table>
   <tr>
         <!-- BEGIN AUSGABE -->
         <td>{NAME}</td>
         <td>{VORNAME}</td>
         <!-- END AUSGABE -->
   </tr>
   </table>

   <form action="formular.php" method="post">
   <table>
   <tr>
         <!-- BEGIN EINGABE -->
         <td><input type="text" name="{INPUTNAME}"
                                value="{INPUTVALUE}"></td>
         <!-- END EINGABE -->
   </tr>
   </table>

   <input type="submit" name="send" value="Abschicken">

   </form>
</body>
</html>
```

Dieses Beispiel entspricht bis auf den hier Fett gedruckten Code dem Beipiel des vorherigen Kapitels. Neu hinzugekommen ist ein kleines, sehr einfaches Formular. Das

Generieren der Formularelemente geschieht im Block EINGABEN. Es ist aus diesem Code noch nicht ersichtlich was und wie viel abgefragt werden soll. Dies ergibt sich erst bei näherem Betrachten des zugehörigen PHP-Codes.

```
<?php
   require_once("HTML/Template/IT.php");

   $template = new HTML_Template_IT("./");

   $template->loadTemplatefile("templates.tpl",true,true);

   // Auswerten der Formulareingaben
   if( isset( $_POST['name'] ) && isset( $_POST['vorname'] ) )
   {
         $template->setCurrentBlock("AUSGABE");
         $template->setVariable("NAME",$_POST['name']);
         $template->setVariable("VORNAME",$_POST['vorname']);
         $template->parseCurrentBlock();
   }

   $name = array("Müller","Schuster","Meier");
   $vorname = array("Hans","Egon","Lisa");

   $i = 0;
   while($i < 3)
   {
         $template->setCurrentBlock("AUSGABE");
         $template->setVariable("NAME",$name[$i]);
         $template->setVariable("VORNAME",$vorname[$i]);
         $template->parseCurrentBlock();
         $i++;
   }

   $template->setCurrentBlock("EINGABE");
   $template->setVariable("INPUTNAME","vorname");
   $template->setVariable("INPUTVALUE","Vorname");
   $template->parseCurrentBlock();

   $template->setCurrentBlock("EINGABE");
   $template->setVariable("INPUTNAME","name");
   $template->setVariable("INPUTVALUE","Nachname");
   $template->parseCurrentBlock();

   $template->show();
?>
```

Auch dieses PHP-Dokument entspricht dem aus dem vorherigem Kapitel. Neu hinzugekommen ist hier der Fett markierte Code. Man sieht, dass zwei Formularelemente angelegt werden. Eins für die Eingabe des Namens und eins für die Eingabe des Nachnamens. Diesen Elementen werden Namen zugewiesen um später die per POST übergebenen Inhalte analysieren zu können.

```
...
$template->setVariable("INPUTNAME","vorname");
...
$template->setVariable("INPUTNAME","name");
...
```

Außerdem erhalten die Eingabefelder einen vordefinierten Eintrag:

```
...
$template->setVariable("INPUTVALUE","Vorname");
...
$template->setVariable("INPUTVALUE","Name");
...
```

Wird ein Name und ein Vorname eingegeben und danach der Button angeklickt werden die Eingaben, zusammen mit den Namen aus den Arrays ausgegeben.

```
...
$template->setCurrentBlock("AUSGABE");
$template->setVariable("NAME",$_POST['name']);
$template->setVariable("VORNAME",$_POST['vorname']);
$template->parseCurrentBlock();
...
```