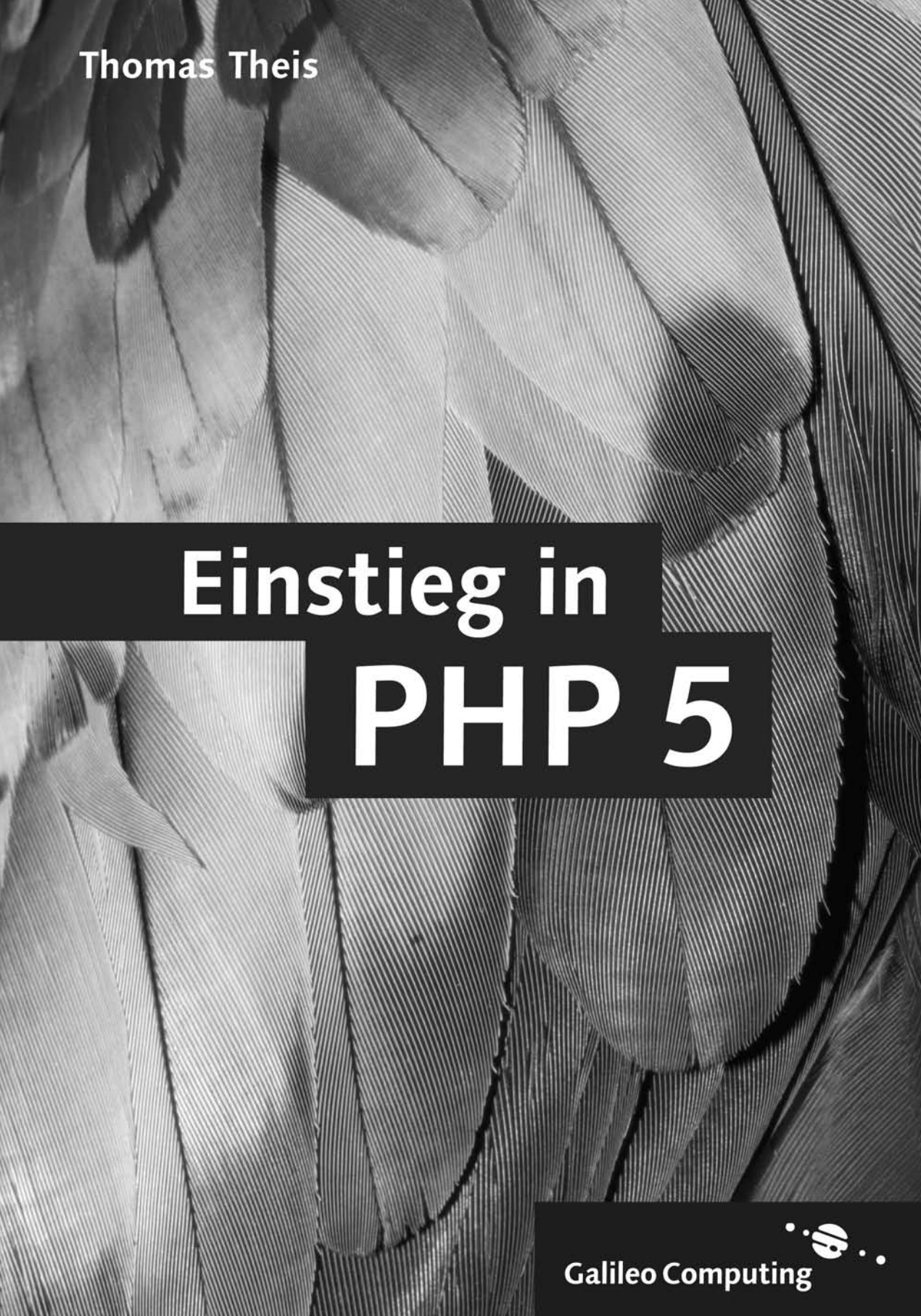

Thomas Theis
Einstieg in
PHP 5
Galileo Computing

Auf einen Blick

Inhalt

B PHP-Programmier-kurs

B PHP-Programmierkurs

In diesem Kapitel werden Sie in die Lage versetzt, erfolgreich Programme in PHP zu schreiben. Sie lernen Variablen und Felder, Operatoren, Kontrollstrukturen und Funktionen kennen. Die Auswertung von Formularen und einige umfangreichere Beispiele runden das Kapitel ab.

Programmierstil

Eine grundsätzliche Bemerkung zum Beginn des Programmierkurses: In diesem Buch sollen nur nicht nur die Kenntnisse der Sprache PHP vermittelt werden, sondern auch ein übersichtlicher, strukturierter Programmierstil. Nach meiner Erfahrung vereinfacht dies sowohl die Arbeit eines einzelnen Entwicklers als auch die Zusammenarbeit eines Entwickler-Teams.

typischer Einsatzzweck

Es wird nicht jede Einzelheit und Komponente der Sprache PHP und ihrer Möglichkeiten erklärt, sondern es werden für viele denkbare Anwendungsfälle jeweils Lösungen angeboten und der typische Einsatzzweck erläutert, ohne durch die Vielfalt zu verwirren.

B.1 Einbettung von PHP in HTML

Grundsätzlich gibt es mehrere Wege, PHP-Programme in HTML-Dateien einzubetten. In den meisten PHP-Programmen wird jedoch die folgende Methode verwendet:

```
<?php
   [PHP-Anweisung]
   [PHP-Anweisung]
   [PHP-Anweisung]
?>
```

<?php ... ?>

Die Markierung `<?php` leitet eine einzelne PHP-Anweisung oder einen Block von PHP-Anweisungen ein. Diese werden bearbeitet bis zur Markierung `?>`, die das Ende des Blocks darstellt.

PHP-Blöcke können sowohl vollständig innerhalb des Dokumentkopfes (`head`) als auch vollständig innerhalb des Dokumentrumpfes (`body`) einer HTML-Seite untergebracht werden. Sie dürfen allerdings nicht im `head` beginnen und erst im `body` enden. Die gesamte Datei wird von oben nach unten abgearbeitet, es kann mehrmals zwischen HTML und PHP gewechselt werden.

HTML-Kurs

Zur Auffrischung beziehungsweise Vertiefung der HTML-Kenntnisse soll an dieser Stelle auf Kapitel I, Anhang, verwiesen werden. Dort finden sich:

- ein Schnellkurs »HTML für PHP«, in dem die wichtigsten HTML-Themen, die zur PHP-Programmierung notwendig sind, erläutert werden,
- ein ausführlicher HTML-Kurs.

Das nachfolgende, vollständige Beispiel verdeutlicht die Einbettung von PHP-Code in HTML:

```
<html>
<head>
<title> Titelzeile der Datei </title>
</head>
<body>
Die erste Zeile in HTML <p>
<?php echo "Die zweite Zeile in PHP<p>"; ?>
Die dritte Zeile in HTML <p>
<?php
   echo "Die vierte Zeile in PHP<p>";
   echo "Die fünfte Zeile in PHP<p>";
?>
</body>
</html>
```

Listing B.1 Datei ub01.php

echo

Die PHP-Anweisung `echo` gibt den angegebenen Text auf dem Bildschirm aus. Der Text muss in Anführungsstrichen geschrieben werden. Falls der Text HTML-Markierungen beinhaltet (hier `<p>` für einen Absatzumbruch), werden diese ausgeführt.

Die Ausgabe des Programms im Browser:

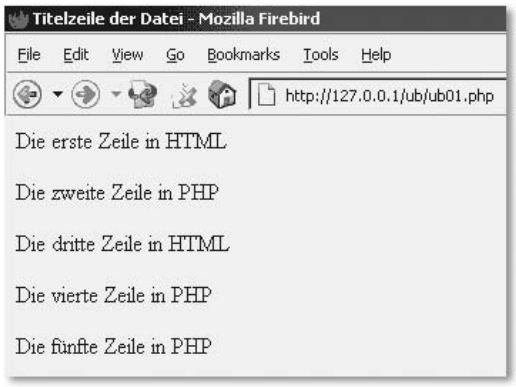


Abbildung B.1 Einbettung von PHP in HTML

B.1.1 Kommentare

Kommentare

Mit Hilfe von Kommentaren wird ein Programm lesbarer. Kommentare werden nicht ausgeführt, sondern dienen nur der Information des Entwicklers, besonders bei umfangreichen Programmen. Sollte es sich um eine Gruppe von Entwicklern handeln oder sollte das Programm später von anderen Entwicklern weiter bearbeitet werden, so ist die Notwendigkeit der Kommentierung der eigenen Programmzeilen für die Kollegen noch größer.

Hinweis: Erfahrungsgemäß gibt es immer wieder Entwickler, die ihre Programme nur minimal kommentieren. Dies stellt sich nach kurzer Zeit als Nachteil für sie selbst und ihre Kollegen heraus.

Man unterscheidet zwischen einzeiligen und mehrzeiligen Kommentaren:

- Ein einzeiliger Kommentar beginnt mit den Zeichen `//` und endet am Ende der Zeile. Er wird im Allgemeinen zur Kommentierung einzelner Begriffe verwendet. **//**
- Ein mehrzeiliger Kommentar beginnt mit den Zeichen `/*` und endet mit den Zeichen `*/`. Er wird üblicherweise zur Erläuterung eines ganzen Programmblocks verwendet. **/* ... */**

Ein Beispiel:

```
<html>
<body>
<?php
```

```
   echo "Das ist der Anfang";     // Kommentar
                                  // bis zum Zeilenende
   /* Ein Kommentar über
      mehrere Zeilen hinweg */
   echo " und hier das Ende des Programms";
?>
</body>
</html>
```

Listing B.2 Datei ub02.php

Die Ausgabe des Programms im Browser:

Abbildung B.2 Programm (ohne sichtbare Kommentare)

Übung UB03

Schreiben Sie ein PHP-Programm innerhalb einer Webseite (Datei `ub03.php`) mit Kommentarzeilen. Speichern Sie die Datei im Hauptverzeichnis Ihres Webservers und testen Sie das Programm, indem Sie einen Browser aufrufen und die passende Adresse eingeben:

- Unter Linux ist das Hauptverzeichnis des Webservers `/usr/local/httpd/htdocs`.
- Unter Windows mit dem Apache Web Server ist das Hauptverzeichnis des Webservers `c:\apache\htdocs`.
- Als Adresse zum Aufruf des oben angegebenen Programms ist in beiden Fällen `http://localhost/ub03.php` einzugeben.

Die Ausgabe des Programms im Browser sollte wie folgt aussehen:

Abbildung B.3 Ergebnis Übung UB03

B.2 Variablen, Datentypen und Operatoren

Innerhalb eines Programms können Informationen zur späteren Verwendung in Variablen gespeichert werden. Diese Variablen unterscheiden sich in ihren Datentypen. PHP unterstützt Datentypen für: Variable

- ganze Zahlen,
- Zahlen mit Nachkommastellen,
- Zeichenketten (Strings),
- Felder (ein- und mehrdimensionale Felder von Variablen),
- Objekte.

Der Datentyp für eine Variable wird nicht vom Programmierer festgelegt, sondern richtet sich nach dem Zusammenhang, in dem die Variable genutzt wird. Eine Variable kann ihren Datentyp innerhalb eines Programms wechseln. Im Unterschied zu vielen anderen Programmiersprachen findet in PHP keine Variablendeklaration statt. Dies bedeutet, dass eine Variable bei ihrem ersten Erscheinen sofort benutzt werden kann und dem Programm nicht vorher bekannt gemacht werden muss. Datentyp

Zunächst geht es um die so genannten »einfachen Datentypen« (Zahlen und Zeichenketten), mit denen viele Aufgaben im Zusammenhang mit der Programmierung bereits erledigt werden können. Später kommen dann die Felder und Objekte hinzu.

B.2.1 Namen

Für die Namen von Variablen (und später auch Funktionen) gelten einige Regeln: Variablennamen

- Sie müssen mit einem Dollarzeichen beginnen. $
- Sie dürfen keine Leerzeichen enthalten.
- Sie dürfen nur aus Buchstaben und Ziffern bestehen, dabei muss das erste Zeichen ein Buchstabe sein; es sind Groß- und Kleinbuchstaben erlaubt. Es wird zwischen Groß- und Kleinschreibung unterschieden (`$HokusPokus` ist nicht das Gleiche wie `$hokuspokus`).
- Sie dürfen keine deutschen Umlaute oder scharfes `ß` enthalten.
- Sie dürfen als einziges Sonderzeichen den Unterstrich »_« enthalten.
- Sie dürfen nicht mit einem reservierten Wort identisch sein, also zum Beispiel mit einem Befehl aus der Sprache PHP.

Man sollte selbsterklärende Namen vergeben. Das hat den Vorteil, dass sich jeder, der sich später mit dem Programm befasst, sofort zurechtfindet. Einige Beispiele: `$Startmeldung`, `$Temperaturwert`, `$XKoordinate`, `$Ywert`

B.2.2 Variablen für Zahlen

Betrachten wir einmal das folgende Programm, in dem der Preis für eine Tankfüllung Benzin berechnet wird:

```
<html>
<body>
<?php
   $liter = 14;
   $preis = 1.15;
   $zahlung = $liter * $preis;
   echo $zahlung;
?>
</body>
</html>
```

Listing B.3 Datei ub04.php

Die Aufgabe dieses Programms ist die Multiplikation von zwei Zahlen und die Ausgabe des Rechenergebnisses. Dies wird wie folgt durchgeführt:

Zahlenvariable

- Die Variable `$liter` wird eingeführt, und es wird ihr der Wert `14` zugewiesen, wodurch `$liter` zu einer Variablen für eine ganze Zahl wird.
- Die Variable `$preis` wird eingeführt, und es wird ihr der Wert `1.15` zugewiesen, also wird `$preis` zu einer Variablen für eine Zahl mit Nachkommastellen (dabei muss ein Punkt als Dezimaltrennzeichen verwendet werden).
- Die Variable `$zahlung` wird eingeführt, `$liter` und `$preis` werden multipliziert und das Ergebnis wird `$zahlung` zugewiesen, damit wurde `$zahlung` ebenfalls zu einer Variablen für eine Zahl mit Nachkommastellen.
- Der Wert von `$zahlung`, also `16.1`, wird ausgegeben.

Die Ausgabe des Programms im Browser:

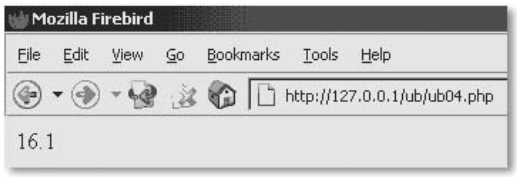


Abbildung B.4 Ergebnis einer einfachen Berechnung

Hinweis: Eine Zahl mit Nachkommastellen kann auch als so genannte Exponentialzahl ausgedrückt werden. Im oben angegebenen Programm hätte man eine der folgenden Schreibweisen verwenden können, dies hätte zum gleichen Ergebnis geführt:

Schreibweise	Berechnung	Ergebnis
`$preis = 0.115e1;`	`0.115 * 101 = 0.115 * 10`	`1.15`
`$preis = 115e-2;`	`115 * 10-2 = 115 * 0.01`	`1.15`

B.2.3 Rechenoperatoren für Zahlen

Bei Zahlen können die folgenden Rechenoperatoren (= arithmetischen Operatoren) verwendet werden:

Rechenoperatoren
+ – * / %

Operator	Bedeutung
+	Addition
–	Subtraktion
*	Multiplikation
/	Division
%	Modulo, Rest bei einer ganzzahligen Division, zum Beispiel ergibt `7%3` den Wert `1`, denn `7` durch `3` ergibt `2 Rest 1`

Ein weiteres Beispiel mit einer etwas umfangreicheren Berechnung:

```
<html>
<body>
<?php
   $liter1 = 16;
   $liter2 = 23;
```

```
   $liter3 = 34;
   $preis = 1.15;
   $gesamtzahlung = ($liter1 + $liter2 + $liter3)
      * $preis;
   echo $gesamtzahlung;
?>
</body>
</html>
```

Listing B.4 Datei ub05.php

Priorität der Operatoren

Es ist zu beachten, dass (wie in der Mathematik üblich) Multiplikation und Division eine höhere Priorität als Addition und Subtraktion haben, also zuerst ausgeführt werden. Außerdem findet bei Berechnungsausdrücken die Bearbeitung von links nach rechts statt. Mit dem Setzen von Klammern kann der Entwickler allerdings die Reihenfolge beeinflussen. Ausdrücke in Klammern werden zuerst vollständig ausgerechnet, das Ergebnis fließt später in die restliche Berechnung ein.

Zum vorliegenden Programm: Die Variablen `$liter1`, `$liter2`, `$liter3` und `$preis` werden eingeführt und mit Werten belegt. Die Variable `$gesamtzahlung` wird wie folgt errechnet:

- Die drei Literzahlen werden addiert (ergibt `73`).
- Die Gesamt-Literzahl wird mit dem Preis multipliziert (ergibt `83.95`).

Die Ausgabe des Programms im Browser:

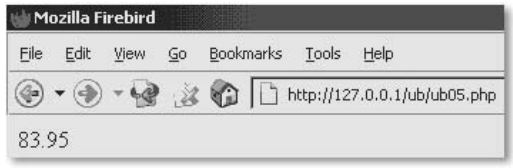


Abbildung B.5 Ergebnis einer umfangreicheren Berechnung

Der Ausdruck `$gesamtzahlung = $liter1 + $liter2 + $liter3 * $preis`, also ohne Klammern, führt nicht zum richtigen Ergebnis, da in diesem Falle:

- die Multiplikation zuerst ausgeführt wird, es ergibt sich der Preis für `34` Liter,
- anschließend zu diesem Preis die beiden anderen Literzahlen addiert werden.

Übung UB06

Berechnen Sie in einem PHP-Programm (Datei `ub06.php`) den Brutto-Preis eines Einkaufs. Es wurden drei Artikel eingekauft, die Netto-Preise der einzelnen Artikel betragen: `22.50 Euro`, `12.30 Euro` und `5.20 Euro`. Der Brutto-Preis berechnet sich bekanntlich aus dem Netto-Preis zuzüglich 16 Prozent Umsatzsteuer. Es muss also der Faktor `1.16` in die Berechnung einbezogen werden. Speichern Sie die Datei im Hauptverzeichnis Ihres Webservers und testen Sie das Programm, indem Sie einen Browser aufrufen und die Adresse `http://localhost/ub06.php` eingeben.

Die Ausgabe des Programms im Browser sollte wie folgt aussehen:

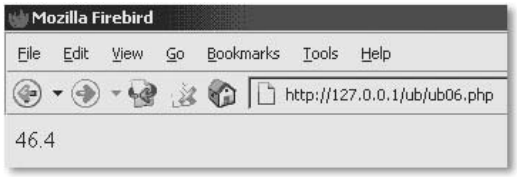


Abbildung B.6 Ergebnis Übung UB06

B.2.4 Variablen und Operatoren für Zeichenketten

Zeichenketten (Strings) müssen in doppelte Hochkommata (`" "`) oder in einfache Hochkommata (`' '`) eingeschlossen werden.

Strings

Das Zeichen `.` (Punkt) dient zur Verkettung mehrerer Zeichenketten miteinander beziehungsweise mehrerer Zahlen und Zeichenketten. Dies wird zum Beispiel für eine kommentierte Ergebnisausgabe genutzt. Der Operator `.=` (Punkt gleich) kann zur Vergrößerung einer Zeichenkette eingesetzt werden. Falls die Zeichenketten HTML-Code enthalten, so gelangt dieser HTML-Code zur Ausführung. Ein Beispielprogramm:

Hochkomma, Punkt

```
<html>
<body>
<?php
$liter = 14;
$preis = 1.15;
$zahlung = $liter * $preis;
$einheit1 = "Liter";
$einheit2 = 'Euro';
$gesamt = "Tankfüllung: " . $liter . " " . $einheit1;
```

```
$gesamt .= " kosten " . $zahlung . " " .
     $einheit2."<p>";
echo $gesamt;
echo "Tankfüllung: $liter $einheit1 kosten $zahlung
     $einheit2<p>";
echo 'Tankfüllung: $liter $einheit1 kosten $zahlung
     $einheit2<p>';?>
</body>
</html>
```

Listing B.5 Datei ub07.php

Zur Erläuterung:

- Im ersten Teil des Programms findet die Berechnung des Preises statt.
- Den Variablen `$einheit1` und `$einheit2` werden Zeichenketten zugewiesen, in doppelten Hochkommata beziehungsweise in einfachen Hochkommata.
- Der Variablen `$gesamt` wird eine Zeichenkette zugewiesen, die sich aus einzelnen Zeichenketten, Zahlenvariablen, Zeichenkettenvariablen und HTML-Code zusammensetzt (Operator `.`).
- Die Zeichenkette `$gesamt` wird verlängert (Operator `.=`).
- Die Zeichenkette `$gesamt` wird ausgegeben.
- Der gleiche Ausgabetext soll auf zwei weitere Arten ausgegeben werden. Der Wert einer einzelnen Variablen wird auch dann ausgegeben, falls die Variable innerhalb einer Zeichenkette untergebracht wurde. Diese Form wird häufig verwendet. Es ist allerdings darauf zu achten, dass die Zeichenkette zwischen doppelte Hochkommata gesetzt wurde.
- Falls die Variable innerhalb einer Zeichenkette mit einfachen Hochkommata steht, wird nur der Name der Variablen, aber nicht der Wert der Variablen im Text ausgegeben, siehe Ausgabe. Das ist normalerweise nicht erwünscht.

Ein Tipp zum besseren Verständnis: Verfolgen Sie jeden einzelnen Schritt des Programms und notieren Sie den aktuellen Wert jeder Variablen, sobald sich dieser ändert.

Hinweis: Beim Schreiben eines Programms im Editor sollte innerhalb einer Zeichenkette, also innerhalb von einfachen oder doppelten Hoch-

kommata, kein Zeilenumbruch erfolgen. In diesem Buch ist dies aber aus drucktechnischen Gründen an einigen Stellen notwendig, einige Zeichenketten sind einfach zu lang. Sie erkennen zusammengehörige, lange Zeichenketten leicht an dem geringeren Abstand zwischen den einzelnen Zeilen und an der Einrückung ab der zweiten Zeile. An diesen Stellen wurde kein Absatzumbruch, sondern ein manueller Zeilenwechsel durchgeführt. Falls Sie die betreffende Programmstelle übernehmen, sollten Sie diese Zeilen unbedingt in einer einzigen Zeile schreiben.

Die Ausgabe des Programms im Browser:

Abbildung B.7 Arbeiten mit Zeichenketten

Übung UB08

Schreiben Sie das Programm aus der vorherigen Übung `UB06` um (Datei `ub08.php`). Zwischenergebnis und Endergebnis sollen errechnet werden. Speichern Sie die Datei im Hauptverzeichnis Ihres Webservers und testen Sie das Programm, indem Sie einen Browser aufrufen und die Adresse `http://localhost/ub08.php` eingeben.

Die Ausgabe des Programms im Browser sollte wie folgt aussehen:

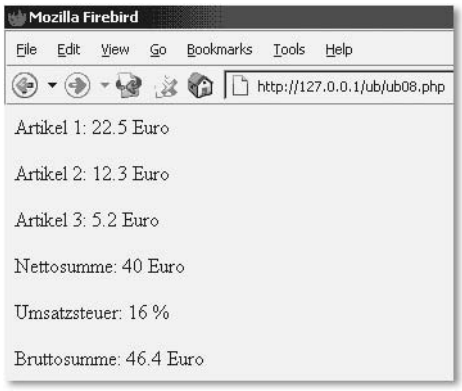


Abbildung B.8 Ergebnis Übung UB08

B.3 Einfache Formularauswertung

Formulare auswerten

In den bisher gezeigten Beispielen hatte der Benutzer eines Programms noch keine Möglichkeit, eigene Eingaben vorzunehmen. Er konnte das Programm nur aufrufen und das Ergebnis betrachten.

Eine besondere Stärke und ein typischer Einsatzzweck von PHP ist jedoch die Auswertung von Benutzereingaben aus Formularen. Durch eine solche Auswertung wird die dynamische Informationsübermittlung zwischen Benutzer und Webserver erst ermöglicht.

Dem Betrachter wird zunächst ein Formular vorgelegt, in dem er eigene Einträge vornehmen kann beziehungsweise bei dem er unter vorgefertigten Einträgen auswählen kann. Er füllt das Formular aus, sendet es ab und erhält nach der Auswertung eine Antwort vom Webserver.

B.3.1 Eingabeformular

Text-Eingabefeld

In diesem Abschnitt soll eine Informationsübermittlung mit Hilfe von einzeiligen Text-Eingabefeldern ermöglicht werden. Formulare können noch aus einer Reihe weiterer Elemente bestehen. Diese werden ausführlich in Kapitel C, Daten senden und auswerten, besprochen.

Der HTML-Programmcode des Formulars:

```
<html>
<body>
Bitte tragen Sie Ihren Vornamen und Ihren Nachnamen ein.<
br>
Senden Sie anschließend das Formular ab.<p>
<form action = "ub09a.php" method = "post">
    <input name = "vor"> Vorname<p>
    <input name = "nach"> Nachname<p>
    <input type = "submit">
    <input type = "reset">
</form>
</body>
</html>
```

Listing B.6 Datei ub09.htm

form, action, method

Innerhalb des HTML-Dokumentes befindet sich ein `form`-Container. Die Markierung `<form>` beinhaltet:

- das Attribut `action`, dieses verweist auf die Datei mit dem PHP-Auswertungsprogramm (hier: `u09a.php`),
- das Attribut `method`, dieses verweist auf die Übermittlungsmethode zum Webserver (hier: `post`).

Der `form`-Container beinhaltet die verschiedenen Formularelemente. Dabei handelt es sich um:

- zwei einzeilige Text-Eingabefelder mit den Namen `vor` beziehungsweise `nach` für die Eintragung von Vornamen beziehungsweise Nachnamen,
- eine Schaltfläche zum Absenden (engl.: `submit`). Bei Betätigung werden die eingetragenen Daten an den Server gesendet und es wird das genannte PHP-Auswertungsprogramm angefordert,

submit, reset

- eine Schaltfläche zum Zurücksetzen (engl.: `reset`) des Formulares. Bei Betätigung wird das Formular wieder in den Anfangszustand versetzt, wie es zum Beispiel bei einer Fehleingabe notwendig werden kann.

Die Ausgabe des Formulars im Browser, mit eingegebenen Beispieldaten:

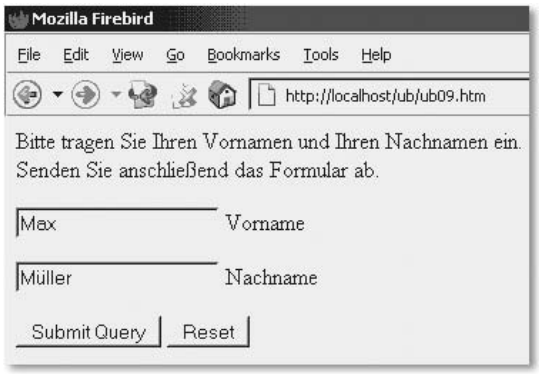


Abbildung B.9 Eingabeformular mit Beispieldaten

Die Auswertung der Eingabedaten kann in PHP grundsätzlich auf zwei verschiedene Arten geschehen, beide werden in den folgenden Abschnitten vorgestellt.

Übung UB10, Teil 1

Erweitern Sie das Beispiel dahingehend, dass eine ganze Adresse eingegeben werden kann (Datei `ub10.htm`). Es soll zusätzlich vier weitere

Eingabefelder für die Angaben Straße, Hausnummer, Postleitzahl und Ort innerhalb des Formulars geben.

Das Formular sollte das folgende Aussehen haben (mit Beispieldaten):

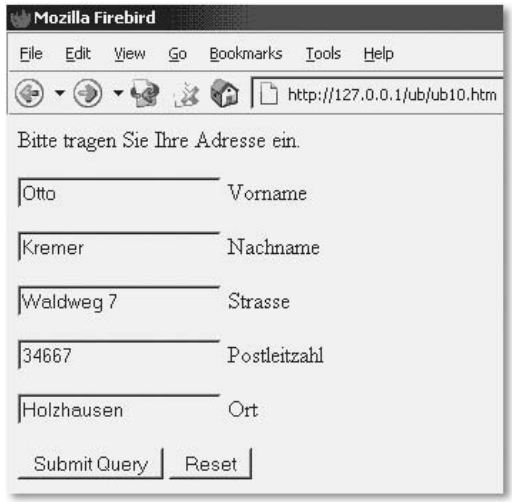


Abbildung B.10 Erweitertes Eingabeformular mit Beispieldaten

B.3.2 Auswertung mit globalen Variablen

globale Variable

Das antwortende PHP-Programm mit globalen Variablen sieht wie folgt aus:

```
<html>
<body>
<?php
   echo "Guten Tag, $vor $nach";
?>
</body>
</html>
```

Listing B.7 Datei ub09a.php

Innerhalb der Antwort-Datei werden die Werte der beiden Variablen `$vor` und `$nach` ausgewertet beziehungsweise ausgegeben. Ihre Werte wurden gemeinsam mit den Namen der beiden Eingabefelder vom aufrufenden Formular aus übermittelt. Aus dem Namen der Eingabefelder werden dabei automatisch globale PHP-Variablen, indem jeweils ein Dollarzeichen davor gesetzt wird. Die Eintragung im Text-Eingabefeld `vor` wird also zum Wert der Variablen `$vor` im Programm.

Falls der Benutzer das oben angegebene Beispiel eingegeben hat, antwortet der Server wie folgt:

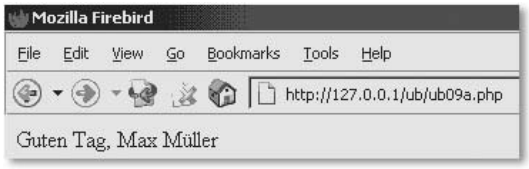


Abbildung B.11 Auswertung der globalen Variablen

Mit dieser Auswertungsmethode lässt sich sehr einfach programmieren. Sie birgt allerdings ein Sicherheitsrisiko, da man einem Benutzer potenziell die Möglichkeit eröffnet, zum Beispiel ausführbaren Code zum Server zu senden, der dort zur Übermittlung von nicht öffentlichen Daten beziehungsweise zur Zerstörung von Daten führen kann.

register globals

Daher wurde ab PHP-Version 4.2.0 der Konfigurationsschalter `register_globals` in der Datei `php.ini` standardmäßig auf den Wert `Off` gestellt. Dies bedeutet, dass die Namen und Werte der Eingabefelder nicht mehr automatisch als globale Variablen im PHP-Programm bekannt sind. Der Betreiber eines Webservers könnte diesen Schalter zwar wieder auf `On` stellen, würde sich damit aber das beschriebene Sicherheitsproblem einhandeln.

B.3.3 Auswertung mit $_POST

$_POST

Man löst das genannte Problem, falls man das superglobale, vordefinierte Feld `$_POST` benutzt. Das antwortende PHP-Programm sieht in diesem Falle wie folgt aus:

```
<html>
<body>
<?php
   echo "Guten Tag, " . $_POST["vor"] .
        " " . $_POST["nach"];
?>
</body>
</html>
```

Listing B.8 Datei ub09b.php

Das Programm erzeugt die gleiche Ausgabe. Falls diese Version aufgerufen werden soll, muss im HTML-Eingabeformular (Datei `u09.htm`)

der Wert des Attributes `action` von `u09a.php` auf `u09b.php` geändert werden!

Es gibt in PHP einige vordefinierte Variablen, unter anderem das assoziative Feld `$_POST`. Aus den Namen der Eingabefelder werden automatisch Elemente dieses Feldes, falls die Übermittlungsmethode `post` verwendet wurde. Diese Elemente können angesprochen werden, indem man ihren Namen in Anführungszeichen und eckigen Klammern hinter dem Namen des Feldes `$_POST` angibt. Die Eintragung im Text-Eingabefeld `vor` wird also zum Wert der Variablen `$_POST["vor"]` im Programm.

Feld-Elemente können allerdings nicht unmittelbar innerhalb einer Zeichenkette ausgegeben werden, wie dies bei einzelnen Variablen der Fall ist. Daher ist die Ausgabezeile mit `echo` etwas umfangreicher. Weitere Einzelheiten zu Feldern und speziell zu assoziativen Feldern in Abschnitt B.6.2, Assoziative Felder.

Im weiteren Verlauf dieses Buches wird mal die eine, mal die andere Methode verwendet. Die Variante in der jeweils anderen Methode kann leicht entwickelt werden. In bestehenden PHP-Projekten, die (noch) nicht umgestellt wurden, beziehungsweise in PHP-Anwendungen für ein Intranet, auf die nur ein begrenzter, kontrollierter Benutzerkreis Zugriff hat, kommt auch die »bequemere« Methode zum Einsatz.

$_GET

Ein Formular kann statt mit der Methode `post` auch mit der Methode `get` versendet werden. Auf die Auswertung mit Hilfe von globalen Variablen hat dies keinen Einfluss. Bei der Auswertung mit Hilfe eines superglobalen, vordefinierten Feldes ist zu beachten, dass das Feld `$_GET` statt des Feldes `$_POST` verwendet werden muss. Die Methode `post` ist im Allgemeinen zu bevorzugen, da sie sicherer und universeller ist.

Übung UB09

Dieses Zusammenspiel von HTML-Datei und PHP-Datei stellt einen wichtigen Schritt dar. Daher zunächst nur eine »einfache« Aufgabe: Geben Sie beide Versionen des angegebenen Beispiels mit Hilfe eines Texteditors ein und speichern Sie es unter den genannten Dateinamen ab (`ub09.htm`, `ub09a.php` und `ub09b.php`). Füllen Sie das Formular aus, senden Sie es in den verschiedenen Versionen ab und kontrollieren Sie die Reaktion des Webservers.

Übung UB10, Teil 2

Erstellen Sie (passend zum Formular aus Übung `UB10`, Teil 1) jeweils ein PHP-Programm für beide Auswertungsmethoden (`ub10a.php` und `ub10b.php`), das die Daten des Benutzers bestätigt. Falls der Benutzer die oben angegebenen Beispieldaten eingegeben hat, sollte die Ausgabe des Programms im Browser wie folgt aussehen:

Abbildung B.12 Auswertung des erweiterten Eingabeformulars

B.3.4 Umwandlung von Zeichenketten in Zahlen

Ein Text-Eingabefeld eines Formulars nimmt eine Zeichenkette auf, und es wird auch eine Zeichenkette an das PHP-Programm übermittelt. Häufig sollen jedoch Zahlen, zum Beispiel zur Ausführung von Berechnungen, übermittelt werden. Dabei sind für den Entwickler die folgenden Regeln zu beachten.

Umwandlung, Konvertierung

Bei der Umwandlung einer Zeichenkette (Konvertierung) ist der Beginn der Zeichenkette wichtig. Falls die Zeichenkette mit gültigen numerischen Zeichen beginnt, so werden diese numerischen Zeichen genutzt. Anderenfalls ergibt sich der Wert 0. Eine gültige Folge von numerischen Zeichen beinhaltet:

- ein Vorzeichen (optional),
- eine oder mehrere Ziffern,
- einen Dezimalpunkt (optional),
- einen Exponenten (optional), der Exponent ist ein kleines »`e`« oder ein großes »`E`«, gefolgt von einer oder mehreren Ziffern.

Die Zeichenkette wird interpretiert als:

- ganze Zahl, falls sie nur Ziffern beinhaltet,
- Zahl mit Nachkommastellen, falls sie neben den Ziffern die Zeichen `.` (Punkt), `e` oder `E` beinhaltet.

Einige Beispiele:

Zeichenkette	Wert	Datentyp
"352"	352	ganze Zahl
"352xz"	352	ganze Zahl
"xz352"	0	Zeichenkette
"35.2"	35.2	Zahl mit Nachkommastellen
"35.2xz"	35.2	Zahl mit Nachkommastellen
"xz35.2"	0	Zeichenkette
"-352"	-352	ganze Zahl
"35e2"	3500	Zahl mit (möglichen) Nachkommastellen
"35e-2"	0.35	Zahl mit Nachkommastellen

Falls man Zeichenkettenvariablen der Sicherheit halber explizit (also vom Programmentwickler gesteuert) in Zahlen umwandeln möchte, kann man die beiden Funktionen `doubleval()` beziehungsweise `intval()` anwenden. Ein kleines Beispiel für zwei Umwandlungen:

```
$a = "435";
$a = intval($a);
$b = "22.6";
$b = doubleval($b);
```

Nach Bearbeitung dieses Programmteiles stehen die Variablen `$a` und `$b` auf jeden Fall als Zahlenvariablen mit dem ganzzahligen Wert `435` beziehungsweise dem Wert `22.6` für weitere Berechnungen zur Verfügung.

In den Einführungsbeispielen dieses Buches werden Eingabefehler des Benutzers nicht immer abgefangen. Die Programme würden sonst unnötig umfangreich und unverständlich. Später wird man in den Programmen Routinen einbauen, die möglichst alle Eingabefehler abfangen, aber es gilt immer der Grundsatz: Kein Programm ist vollständig gegen Eingabefehler gesichert.

Im nachfolgenden Beispiel wird der Benutzer aufgefordert, in einem Formular zwei Zahlen einzugeben und das Formular abzusenden. Ein PHP-Programm berechnet die Summe dieser beiden Zahlen und gibt das Ergebnis bekannt.

Der HTML-Code des Formulars:

```
<html>
<body>
Bitte tragen Sie zwei Zahlen ein und senden Sie das
   Formular ab.<p>
<form action = "ub11a.php" method = "post">
    Wert 1: <input name = "w1"><p>
    Wert 2: <input name = "w2"><p>
    <input type = "submit">
    <input type = "reset">
</form>
</body>
</html>
```

Listing B.9 Datei ub11.htm

Die beiden Versionen des PHP-Programms:

```
<html>
<body>
<?php
   $erg = $w1 + $w2;
   echo "Die Summe von $w1 und $w2 ist $erg";
?>
</body>
</html>
```

Listing B.10 Datei ub11a.php

```
<html>
<body>
<?php
   $erg = $_POST["w1"] + $_POST["w2"];
   echo "Die Summe von " . $_POST["w1"] . " und " .
      $_POST["w2"] . " ist $erg";
?>
</body>
</html>
```

Listing B.11 Datei ub11b.php

Ein Aufruf mit folgenden Eingabewerten:

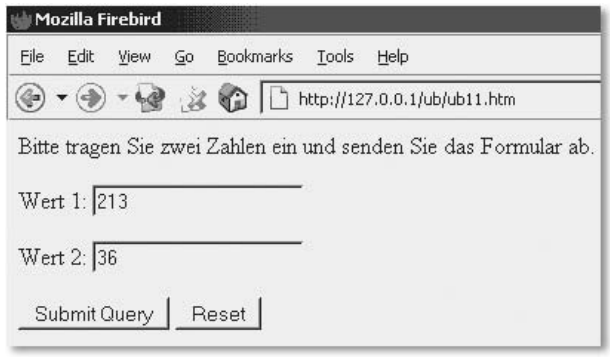


Abbildung B.13 Senden von Zahlen

ergibt in beiden Fällen die Antwort:

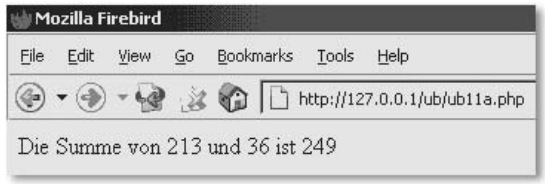


Abbildung B.14 Umwandlung und Berechnung des Ergebnisses

In beiden Versionen des Antwort-Programms werden die eingegebenen Zeichenketten nach den oben angegebenen Regeln automatisch in Zahlen umgewandelt.

Übung UB12

Erstellen Sie ein Eingabeformular (Datei `ub12.htm`) und ein dazu passendes PHP-Programm (Datei `ub12.php`), mit dessen Hilfe das Quadrat einer Zahl berechnet werden kann. Die Zahl soll also mit sich selbst multipliziert werden. Wählen Sie eine der beiden genannten Methoden zur Auswertung der Eingabedaten.

Formular und Ergebnis sollten wie folgt aussehen:

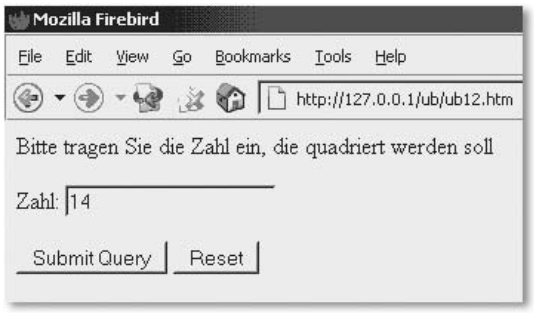


Abbildung B.15 Eingabe Übung 12

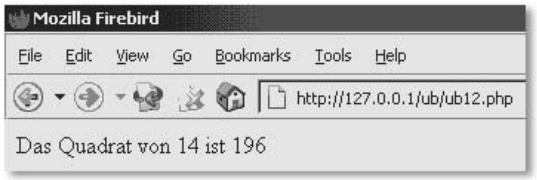


Abbildung B.16 Ergebnis Übung 12

B.4 Verzweigungen

Bisher wurden die Dateien mit dem HTML-Code und dem PHP-Code rein sequenziell abgearbeitet, das heißt, eine Anweisung nach der anderen wurde durchgeführt. Tatsächlich sind aber Programme auch in der Lage, auf unterschiedliche Bedingungen zu reagieren. Einzelne Anweisungen werden nur in bestimmten Situationen ausgeführt.

if, if-else

Die Ausführung dieser Anweisungen wird in solchen Fällen von einer oder mehreren Bedingungen abhängig gemacht (`if`-Anweisung). Je nachdem, ob die Bedingung zutrifft, werden die entsprechenden Anweisungen ausgeführt oder nicht. Darüber hinaus können bei Nichterfüllung der Bedingung alternative Anweisungen bearbeitet werden (`if-else`-Anweisung). Man nennt diese Stellen in Programmen »Verzweigungen«.

Bedingung, wahr, falsch

Bedingungen werden mit Hilfe von Wahrheitswerten (wahr oder falsch beziehungsweise `true` oder `false`) und Vergleichsoperatoren erstellt. Eine Tabelle der Vergleichsoperatoren schließt sich an.

Operator	Bedeutung	gilt
==	Gleichheit	für Zahlen und Zeichenketten
!=	Ungleichheit	für Zahlen und Zeichenketten
>	größer als	nur für Zahlen
<	kleiner als	nur für Zahlen
>=	größer oder gleich	nur für Zahlen
<=	kleiner oder gleich	nur für Zahlen

Vergleichsoperatoren

Bei der Überprüfung auf Gleichheit ist besonders auf das doppelte Gleichheitszeichen zu achten. Es handelt sich um eine Bedingung und nicht um eine Zuweisung.

B.4.1 if-Anweisung

Ein Beispiel für eine Verzweigung mit der `if`-Anweisung:

```
<html>
<body>
<?php
   $preis = 0.98;
   if ($preis < 1) echo "Der Preis liegt unter 1 Euro";
?>
</body>
</html>
```

Listing B.12 Datei ub13.php

Falls `$preis` kleiner als `1` ist, wird der entsprechende Text in das Dokument geschrieben, anderenfalls geschieht nichts. Die Bedingung (`$preis < 1`) muss in Klammern stehen.

Die Ausgabe sieht wie folgt aus:

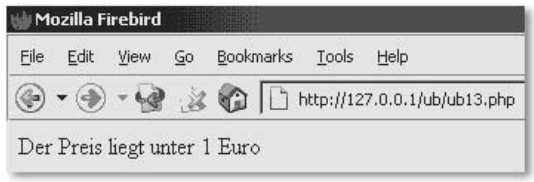


Abbildung B.17 Einfache if-Bedingung

Ein weiteres Beispiel:

```
<html>
<body>
<?php
   $preis = 0.98;
   if ($preis < 1)
   {
      echo "Der Preis liegt unter 1 Euro<br>";
      echo "Das ist günstig";
   }
?>
</body>
</html>
```

Listing B.13 Datei ub14.php

Falls aufgrund einer Bedingung mehrere Anweisungen ausgeführt werden sollen, so müssen sie innerhalb von geschweiften Klammern stehen. Dies nennt man einen Anweisungsblock. In diesem Programm werden zwei Ausgaben erzeugt, da `$preis` kleiner als 1 ist.

Anweisungsblock { . . }

Die Ausgabe sieht wie folgt aus:

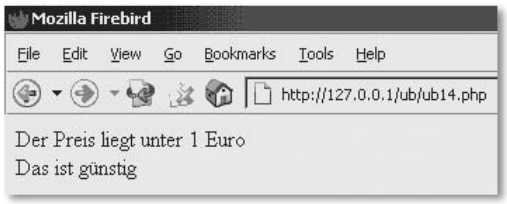


Abbildung B.18 Verzweigung mit Anweisungsblock

B.4.2 if-else-Anweisung

Ein Beispiel für eine Verzweigung mit der `if-else`-Anweisung:

```
<html>
<body>
<?php
   $preis = 1.02;
   if ($preis < 1)
   {
      echo "Der Preis liegt unter 1 Euro<br>";
      echo "Das ist günstig";
```

```
    }
    else
    {
        echo "Der Preis liegt bei 1 Euro oder mehr<br>";
        echo "Langsam wird es teuer";
    }
?>
</body>
</html>
```

Listing B.14 Datei ub15.php

Falls die Bedingung hinter dem `if` nicht zutrifft, werden die Anweisungen hinter dem `else` ausgeführt. Auch hier gilt, dass bei mehreren Anweisungen geschweifte Klammern benutzt werden müssen.

Die Ausgabe sieht wie folgt aus:

Abbildung B.19 Verzweigung mit else

Passwort

Ein weiteres Beispiel (mit Eingabeformular) verdeutlicht den Vergleich von Zeichenketten bei einer Bedingung. Der Benutzer soll ein Zugangs-Passwort eintragen (ausnahmsweise in sichtbarer Form). Das PHP-Programm vergleicht die Eingabe mit dem gespeicherten Passwort und reagiert entsprechend.

Der HTML-Code des Formulars:

```
<html>
<body>
Bitte tragen Sie das Zugangs-Passwort ein<br>
<form action = "ub16.php" method = "post">
    <input name = "pw"><p>
    <input type = "submit">
    <input type = "reset">
</form>
```

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

Listing B.15 Datei ub16.htm

Das Auswertungsprogramm:

```
<html>
<body>
<?php
   if ($pw == "bingo")
      echo "Zugang gestattet";
   else
      echo "Zugang verweigert";
?>
</body>
</html>
```

Listing B.16 Datei ub16.php

Falls der Benutzer Folgendes eingibt:

Abbildung B.20 Eingabe Passwort

bekommt er Zugang, anderenfalls nicht:

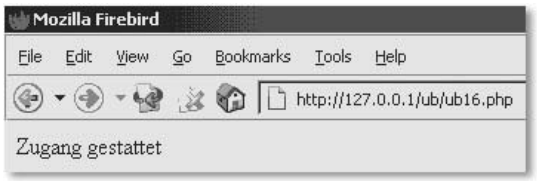


Abbildung B.21 Auswertung der Verzweigung

Übung UB17

Erstellen Sie ein Eingabeformular (Datei `ub17.htm`) und ein dazu passendes PHP-Programm (Datei `ub17.php`). Es soll der Preis für eine Tankfüllung berechnet werden. Es gibt zwei Sorten Benzin: Normal (Preis: `1.05 Euro`) und Super (Preis: `1.15 Euro`).

Der Benutzer gibt im ersten Eingabefeld die getankte Literzahl und im zweiten Eingabefeld entweder ein `N` oder ein `S` ein. Das PHP-Programm ermittelt abhängig von der Sorte und der Menge den zu zahlenden Betrag. Es wird davon ausgegangen, dass der Benutzer keine Fehleingaben macht.

Falls der Benutzer `15` Liter Super tankt:

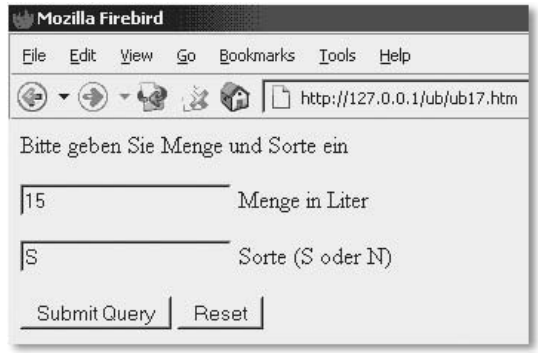


Abbildung B.22 Eingabe Tankvorgang

sollte die Ausgabe des Programms wie folgt aussehen:

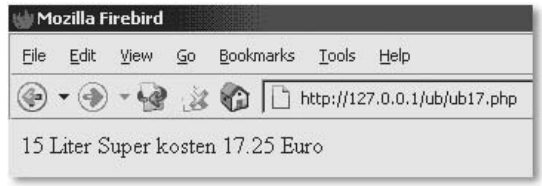


Abbildung B.23 Ergebnis Tankvorgang

Übung UB18

Erweitern Sie die vorherige Übung. Großkunden, die `100` Liter oder mehr tanken, bekommen unabhängig von der Sorte bei dieser Tankstelle `2 Prozent` Rabatt. Benutzen Sie zur Auswertung der Benutzereingaben keine globalen Variablen, sondern das Feld `$_POST`.

Falls der Benutzer `120` Liter Normal tankt:

Abbildung B.24 Eingabe Übung 18

sollte die Ausgabe des Programms wie folgt aussehen:

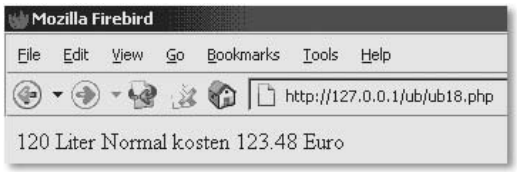


Abbildung B.25 Ergebnis Übung 18

B.4.3 Logische Operatoren

Logische Operatoren dienen zur Verknüpfung von mehreren Bedingungen.

Logische Operatoren

Logisches ODER

Das logische ODER (Zeichen `||`) wird verwendet, falls nur eine von mehreren Bedingungen zutreffen muss. Zur Verdeutlichung wird das Beispiel mit der Passworteingabe (`ub16.htm` und `ub16.php`) erweitert. Es gibt nun zwei Passwörter, die zum erfolgreichen Zugang führen. Das Eingabeformular bleibt gleich, das Auswertungsprogramm sieht wie folgt aus:

logisches oder, ||

```
<html>
<body>
<?php
   if ($pw == "bingo" || $pw == "kuckuck")
      echo "Zugang gestattet";
   else
```

```
      echo "Zugang verweigert";
?>
</body>
</html>
```

Listing B.17 Datei ub19.php

Es gibt zwei Bedingungen, von denen eine zutreffen muss, damit der Zugang gestattet wird. Jede Bedingung muss vollständig formuliert werden. Der Ausdruck `$pw == "bingo" || "kuckuck"` würde zu einer Fehlermeldung führen, da die zweite Bedingung unvollständig ist.

Logisches UND

logisches und, &&

Das logische UND (Zeichen `&&`) wird verwendet, falls alle Bedingungen zutreffen müssen. Dies wird wiederum an einem erweiterten Beispiel der Passworteingabe verdeutlicht. Der Benutzer muss Namen und Passwort eingeben. Der Zugang wird nur gestattet, falls beide richtig sind, es sich also um einen berechtigten und bekannten Benutzer handelt. Zunächst das geänderte Eingabeformular:

```
<html>
<body>
Bitte tragen Sie Namen und Zugangs-Passwort ein<p>
<form action = "ub20.php" method = "post">
    <input name = "bname"> Name<p>
    <input name = "pw"> Passwort<p>
    <input type = "submit">
    <input type = "reset">
</form>
</body>
</html>
```

Listing B.18 Datei ub20.htm

Das Auswertungsprogramm sieht wie folgt aus:

```
<html>
<body>
<?php
   if ($bname == "Maier" && $pw == "kuckuck")
      echo "Zugang gestattet";
   else
      echo "Zugang verweigert";
```

```
?>
</body>
</html>
```

Listing B.19 Datei ub20.php

Gibt der Benutzer zwar den Namen `Maier`, aber ein falsches Passwort ein, so wird der Zugang verweigert, da beides stimmen muss. Das Gleiche trifft zu, falls er den Namen `Meier` (mit e) und das Passwort `kuckuck` eingibt, da in diesem Falle nur die zweite Bedingung zutrifft, siehe Formular und Ausgabe:

Abbildung B.26 Eingabe von Namen und Passwort

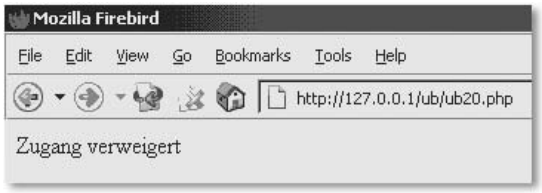


Abbildung B.27 Richtiges Passwort, falscher Name

Logisches NICHT

Mit Hilfe des Operators NICHT (Zeichen `!`) kann man den Wahrheitswert von Bedingungen umkehren. Dies kann bei komplexeren logischen Verknüpfungen helfen.

logisches Nicht, !

Übung UB20

Testen Sie die Beispiele `UB19` und `UB20` mit verschiedenen Passwörtern beziehungsweise Namens- und Passwort-Kombinationen.

B.4.4 Rangordnung der Operatoren

Rangordnung der Operatoren

Ausdrücke mit mehreren Operatoren werden von links nach rechts aufgelöst unter Beachtung der Rangordnung. Nachfolgend eine Tabelle mit der Rangordnung der bisher verwendeten Operatoren. Es wird mit der höchsten Stelle der Rangordnung begonnen.

Operator	Bedeutung
`()`	Klammern
`! -`	logisches NICHT, negatives Vorzeichen
`* / %`	Multiplikation, Division, Modulo
`+ -`	Addition, Subtraktion
`< <= > >=`	kleiner, kleiner oder gleich, größer, größer oder gleich
`== !=`	gleich, ungleich
`&&`	logisches UND
`||`	logisches ODER
`=`	Zuweisung

Klammern stehen an der höchsten Stelle der Rangordnung. Mit ihrer Hilfe kann man die Rangfolge bei den Operatoren beeinflussen und Ausdrücke so bewerten, wie man es wünscht. Zusätzlich kann man Klammern verwenden, falls man sich bei der Rangfolge momentan nicht sicher ist.

Übung UB21

Erweitern Sie das Beispielprogramm aus dem vorherigen Abschnitt. Nur die beiden Benutzer `Marten` (Passwort `Hamburg`) und `Schmitz` (Passwort `Berlin`) sollen Zugang haben (Dateien `ub21.htm` und `ub21.php`). Benutzen Sie zur Auswertung der Benutzereingaben keine globalen Variablen, sondern das Feld `$_POST`.

B.4.5 Mehrfache Verzweigung

mehrfache Verzweigung

Verzweigungen mit `if` und `else` lassen sich auch verschachteln, so dass eine mehrfache Verzweigung (für mehr als zwei Fälle) möglich wird. Ein Beispiel:

```
<html>
<body>
```

```
<?php
   $preis = 1.12;
   if ($preis < 1)
   {
      echo "Der Preis liegt unter 1 Euro<br>";
      echo "Das ist günstig";
   }
   else
   {
      if ($preis <= 1.2)
      {
         echo "Der Preis liegt zw. 1 und 1.2 Euro<br>";
         echo "Langsam wird es teuer";
      }
      else
      {
         echo "Der Preis liegt über 1.20 Euro<br>";
         echo "Das ist viel zu teuer";
      }
   }
?>
</body>
</html>
```

Listing B.20 Datei ub22.php

Falls `$preis` kleiner als `1` ist, trifft die erste Bedingung zu. Die restlichen Bedingungen müssen dann nicht mehr geprüft werden. Falls nicht, kann `$preis` nur noch größer oder gleich `1` sein. Es wird die nächste Bedingung (`$preis <= 1.2`) geprüft. Falls diese auch nicht zutrifft, kann `$preis` nur noch größer als `1.2` sein.

Die Ausgabe sieht wie folgt aus:

Abbildung B.28 Ergebnis mehrfacher Verzweigung

Übung UB23

Erweitern Sie das Programm aus Übung `UB17`. Es soll der Preis für eine Tankfüllung berechnet werden, ohne Rabatt für Großkunden. Es gibt drei Sorten Benzin: Normal (Preis: `1.05 Euro`), Super (Preis: `1.15 Euro`) und Diesel (Preis: `0.90 Euro`).

Der Benutzer gibt im ersten Eingabefeld die getankte Literzahl und im zweiten Eingabefeld entweder ein `N`, ein `S` oder ein `D` ein. Das PHP-Programm ermittelt in Abhängigkeit von der Sorte und der Menge den zu zahlenden Betrag. Es wird davon ausgegangen, dass der Benutzer keine Fehleingaben macht.

Falls der Benutzer `35` Liter Diesel tankt:

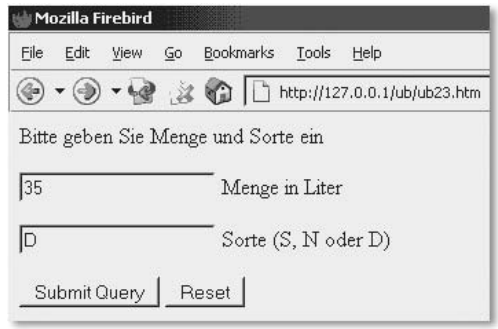


Abbildung B.29 Eingabe Übung UB23

sollte die Ausgabe wie folgt aussehen:

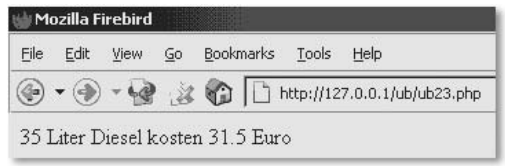


Abbildung B.30 Ergebnis Übung UB23

B.4.6 switch-case-Anweisung

switch-case

Die `switch-case`-Anweisung bietet für einen bestimmten Typ von mehrfachen Verzweigungen eine andere Schreibweise. Sie kann eingesetzt werden, falls die gleiche Variable auf mehrere, feste Werte geprüft werden soll. Diese Form der mehrfachen Verzweigung ist besonders bei vielen verschiedenen Fällen übersichtlicher als eine geschachtelte Verzweigung.

Ein Beispiel bietet die Übungsaufgabe UB23 (siehe oben). Das Eingabeformular aus Datei ub23.htm kann übernommen werden (in ub24.htm). Das Auswertungsprogramm mit switch-case sieht wie folgt aus:

```
<html>
<body>
<?php
   switch($sorte)
   {
      case "N":
         $zahlung = $liter * 1.05;
         echo "$liter L Normal kosten $zahlung Euro";
         break;
      case "S":
         $zahlung = $liter * 1.15;
         echo "$liter L Super kosten $zahlung Euro";
         break;
      case "D":
         $zahlung = $liter * 0.9;
         echo "$liter L Diesel kosten $zahlung Euro";
         break;
      default:
         echo "Als Sorte nur N, S oder D eingeben!";
   }
?>
</body>
</html>
```

Listing B.21 Datei ub24.php

break, default

Es wird ein so genannter switch-Block erzeugt. Innerhalb dieses switch-Blocks wird der Wert der Variablen $sorte untersucht. Die vorhandenen Fälle (engl.: case) werden der Reihe nach verglichen. Sobald einer der Fälle zutrifft, werden alle weiteren Anweisungen bearbeitet, bis man auf die Anweisung break trifft. Die Anweisungen nach dem break werden nicht mehr ausgeführt.

Optional kann die Anweisung default benutzt werden. Diese ist dann nützlich, falls keiner der genannten Fälle zutraf. Dies wäre im oben angegebenen Programm der Fall, falls der Benutzer als Sorte weder N noch S noch D eingibt.

Falls der Benutzer die Eingaben `15` und `P` macht:

Abbildung B.31 Eingabe für switch-case

so ergibt sich die Ausgabe:

Abbildung B.32 Default-Fall

B.4.7 HTML in Verzweigungsblöcken

HTML und PHP gemischt

Falls innerhalb einer einfachen oder mehrfachen Verzweigung jeweils nur reiner HTML-Code ohne PHP-Variablen ausgegeben werden muss, so ist eine gemischte Schreibweise mit PHP und HTML recht nützlich. Ein Beispiel:

```
<html>
<body>
<?php
$preis = 1.12;
if ($preis < 1):
?>
Der Preis liegt unter 1 Euro<br>
Das ist günstig
<?php else: ?>
Der Preis liegt bei 1 Euro oder darüber<br>
Langsam wird es teuer
<?php endif; ?>
```

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

Listing B.22 Datei ub25.php

Der Ablauf der Verzweigung wird auf mehrere PHP-Blöcke verteilt. Dazwischen kann der HTML-Code ohne `echo`, Anführungsstriche, Semikolon usw. notiert werden.

- Nach der Bedingung `if ($preis < 1)` wird ein Doppelpunkt notiert. Dies bedeutet, dass die Verzweigung noch »offen« ist. Der anschließende HTML-Code bis zum nächsten Teil der Verzweigung wird nur ausgeführt, falls die Bedingung zutrifft.
- Es folgt die `else`-Anweisung, ebenfalls mit einem Doppelpunkt. Für den darauf folgenden HTML-Code gilt das Gleiche. Die Verzweigung ist nach wie vor »offen«.
- Sie wird erst durch die Anweisung `endif` abgeschlossen.

Diese gemischte Schreibweise aus PHP und HTML kann auch für andere Formen der Verzweigung und andere Kontrollstrukturen angewendet werden.

B.5 Schleifen

Schleife, Wiederholung

Falls sich innerhalb eines Programms einzelne Anweisungen oder Blöcke von Anweisungen wiederholen, werden Schleifen verwendet. In PHP (wie in jeder anderen Programmiersprache) gibt es dafür grundsätzlich zwei Möglichkeiten. Welche Variante bei der Lösung des vorliegenden Problems die richtige ist, kann man leicht entscheiden.

- Man verwendet die `for`-Schleife, falls die Anzahl der Wiederholungen dem Programmierer bekannt ist oder sie sich eindeutig im Verlauf des Programms vor der Schleife ergibt (Zählschleife).
- Man verwendet die `while`-Schleife, falls die Anzahl der Wiederholungen dem Programmierer nicht bekannt ist und sie sich nicht eindeutig im Verlauf des Programms vor der Schleife ergibt. Die Wiederholung oder der Abbruch der Schleife ergeben sich erst zur Laufzeit des Programms (bedingungsgesteuerte Schleife).

B.5.1 for-Schleife

for

Die `for`-Schleife wird dazu benutzt, eine feste Anzahl an Wiederholungen zu erzeugen. Entweder ist die Anzahl vorher bekannt oder Start

und Ende der Wiederholung sind bekannt beziehungsweise können errechnet werden. Ein Beispiel:

```
<html>
<body>
<?php
   for ($i=1; $i<=5; $i++)
   {
      echo "Zeile $i <p>";
   }
?>
</body>
</html>
```

Listing B.23 Datei ub26.php

Mit Hilfe des Programms werden fünf Zeilen in das Dokument geschrieben, jeweils mit dem Inhalt `Zeile: <Nummer>`. Die Ausgabe sieht wie folgt aus:

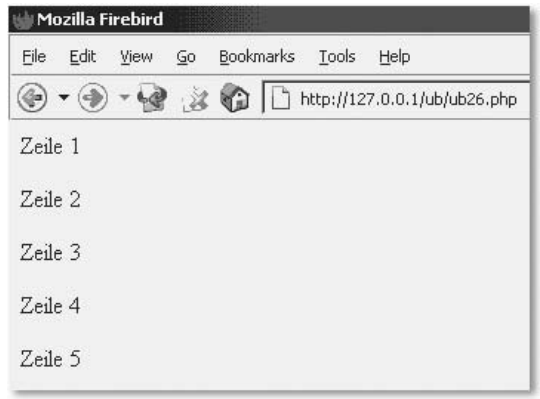


Abbildung B.33 Schleife

Die `for`-Schleife besteht aus Kopf und Rumpf. Der Kopf der `for`-Schleife besteht aus drei Teilen, die durch Semikola voneinander getrennt sind:

- Startwert,
- Bedingung zur Wiederholung,
- Veränderung der Schleifenvariablen.

In diesem Beispiel wird die Variable `$i` als so genannte »Schleifenvariable« verwendet, das heißt, mit Hilfe von `$i` wird die Schleife gesteuert. `$i` bekommt zunächst den Wert `1`. Es wird geprüft, ob die Bedingung zur Wiederholung erfüllt ist. Ist dies der Fall, wird mit dem Anfangswert der Rumpf der Schleife durchlaufen, es ergibt sich also die Ausgabe: `Zeile 1`. Anschließend wird die Variable durch die Veränderung der Schleifenvariablen erhöht (auf `2`).

Es wird geprüft, ob die Bedingung zur Wiederholung noch erfüllt ist. Ist dies der Fall, wird der Rumpf der Schleife mit dem Wert `$i` (Ausgabe: `Zeile 2`) durchlaufen usw. Nach dem `fünften` Durchlauf wird `$i` auf `6` erhöht. Damit trifft die Bedingung zur Wiederholung nicht mehr zu, das Programm beendet die Schleife und läuft dahinter weiter. Im vorliegenden Programm ist dann das Ende erreicht.

Anmerkung: `$i++` ist eine Kurzform der Zuweisung `$i=$i+1`. Häufig verwendet wird auch `$i--`, dies ist eine Kurzform der Zuweisung `$i=$i-1`, also eine Verminderung von `$i` um `1`. ++, --

Auch bei Schleifen gilt: Falls sich die Schleife auf mehrere Anweisungen bezieht, müssen diese in geschweifte Klammern gesetzt werden. Streng genommen wäre dies also beim oben genannten Beispiel nicht notwendig gewesen, aber es schadet auch nicht.

B.5.2 Beispiele für for-Schleifen

Einige Beispiele für Schleifensteuerungen:

Kopf der for-Schleife	`$i` bekommt nacheinander die Werte
`for ($i=10; $i<=15; $i++)`	`10, 11, 12, 13, 14, 15`
`for ($i=10; $i<15; $i++)`	`10, 11, 12, 13, 14`
`for ($i=10; $i>=5; $i--)`	`10, 9, 8, 7, 6, 5`
`for ($i=10; $i>5; $i--)`	`10, 9, 8, 7, 6`
`for ($i=3; $i<=22; $i=$i+3)`	`3, 6, 9, 12, 15, 18, 21`
`for ($i=32; $i>12; $i=$i-4)`	`32, 28, 24, 20, 16`
`for ($i=12; $i<13; $i=$i+0.2)`	`12.0, 12.2, 12.4, 12.6, 12.8`
`$a=6, $b=16, $c=2;` `for ($i=$a; $i<$b; $i=$i+$c)`	`6, 8, 10, 12, 14`

Man sollte immer darauf achten, dass nicht aus Versehen eine Endlosschleife erzeugt wird. Dies könnte man zum Beispiel mit dem folgenden Schleifenkopf erreichen: `for ($i=3; $i>2; $i=$i+3)`. Die Bedingung `$i>2` ist für alle Zahlen, die erzeugt werden, erfüllt. Demnach wird diese Schleife niemals beendet, und das Programm »hängt sich auf«.

Übung UB27

Schreiben Sie ein Programm (Datei `ub27.php`), in dem mit Hilfe von mehreren `for`-Schleifen die nachfolgend angegebenen Zeilen ausgegeben werden. Ein Tipp: Für die letzte Zahlenreihe wird zusätzlich eine `if`-Bedingung benötigt.

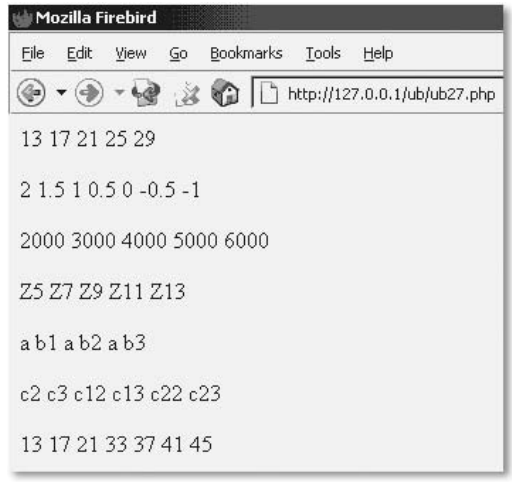


Abbildung B.34 Übung UB 27

B.5.3 Geschachtelte for-Schleifen

geschachtelte Schleife

Schleifen können geschachtelt werden. Dabei befindet sich eine Schleife innerhalb einer anderen Schleife (Schachtelung). Dadurch wird später die Bearbeitung einer zweidimensionalen Struktur wie zum Beispiel einer Tabelle (siehe HTML) oder eines zweidimensionalen Feldes (siehe ein- und mehrdimensionale Felder) möglich. Ein Beispiel:

```
<html>
<body>
<?php
   for ($z=1; $z<=5; $z=$z+1)
   {
```

```
        for ($s=1; $s<=3; $s=$s+1)
        {
           echo "Ze$z/Sp$s ";
        }
        echo "<p>";
    }
?>
</body>
</html>
```

Listing B.24 Datei ub28.php

Die erste (äußere) Schleife wird fünfmal durchlaufen. Innerhalb dieser Schleife steht wiederum eine (innere) Schleife, die bei jedem Durchlauf der äußeren Schleife dreimal durchlaufen wird. Anschließend wird ein Umbruch erzeugt. Es gibt insgesamt `5` mal `3` = `15` Wiederholungen. Die Programmausgabe sieht wie folgt aus:

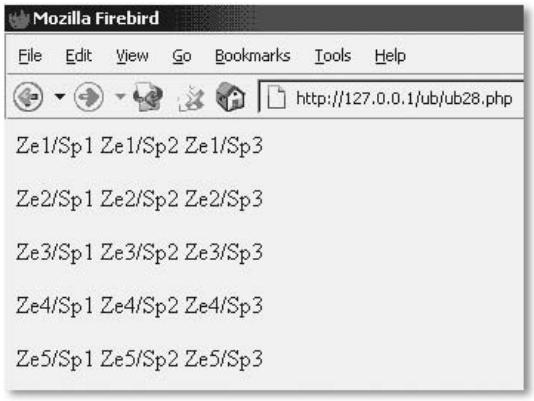


Abbildung B.35 Geschachtelte Schleife

Übung UB29

Schreiben Sie ein Programm (Datei `ub29.php`), in dem mit Hilfe von zwei geschachtelten `for`-Schleifen das kleine Einmaleins ausgegeben wird. Die Ausgabe soll wie folgt aussehen:

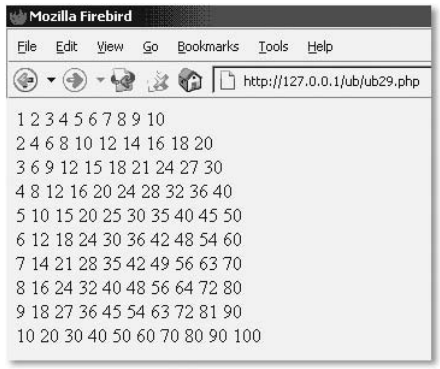


Abbildung B.36 »Kleines Einmaleins«

B.5.4 Schleifen und Tabellen

Schleife mit HTML-Tabelle

Schleifen werden häufig im Zusammenhang mit HTML-Tabellen eingesetzt. Das erweiterte Beispiel aus Datei `ub26.php` kann innerhalb einer Tabellenstruktur zum Beispiel wie folgt angegeben werden:

```
<html>
<body>
<table border>
<?php
   for ($i=8; $i<=15; $i++)
   {
      echo "<tr> <td>Zeile</td>
         <td align='right'>$i</td> </tr>";
   }
?>
</table>
</body>
</html>
```

Listing B.25 Datei ub30.php

Tabellenbeginn und -ende werden hier im HTML-Bereich angegeben. Die veränderlichen Bestandteile (Anzahl der Zeilen und Inhalt der zweiten Spalte) werden im PHP-Bereich angegeben. Bei jedem Durchlauf der Schleife wird eine Tabellenzeile mit jeweils zwei Zellen ausgegeben.

Hinweis: Die Ausrichtung der Zellen (`align='right'`) muss innerhalb der Zeichenkette (die zwischen doppelten Hochkommata steht) zwi-

schen einfachen Hochkommata angegeben werden, da ansonsten für PHP die Zeichenkette zu früh beendet wird.

Die Ausgabe sieht wie folgt aus:

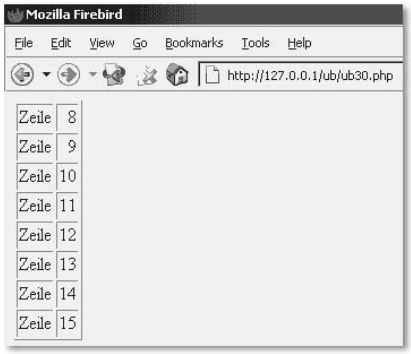


Abbildung B.37 Schleife und Tabelle

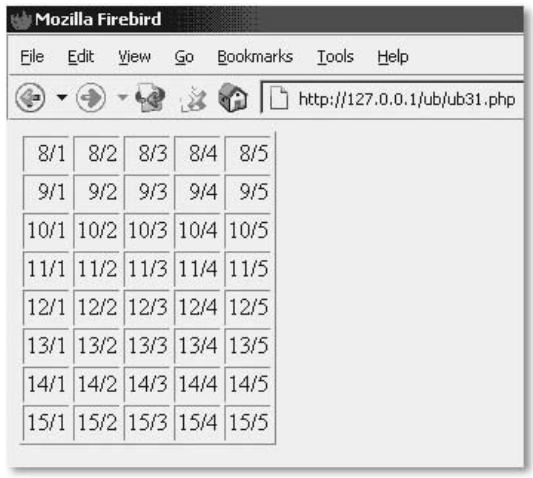
Das erweiterte Beispiel aus Datei `ub28.php` mit einer geschachtelten Schleife innerhalb einer Tabellenstruktur:

```
<html>
<body>
<table border>
<?php
   for ($z=8; $z<=15; $z=$z+1)
   {
      echo "<tr>";
      for ($s=1; $s<=5; $s=$s+1)
      {
         echo "<td align='right'>$z/$s</td>";
      }
      echo "</tr>";
   }
?>
</table>
</body>
</html>
```

Listing B.26 Datei ub31.php

Tabellenbeginn und -ende werden hier wiederum im HTML-Bereich angegeben. Die äußere Schleife sorgt für die Erzeugung der Tabellenzeilen, die innere Schleife für die Erzeugung und Füllung der Zellen.

Die Ausgabe sieht wie folgt aus:

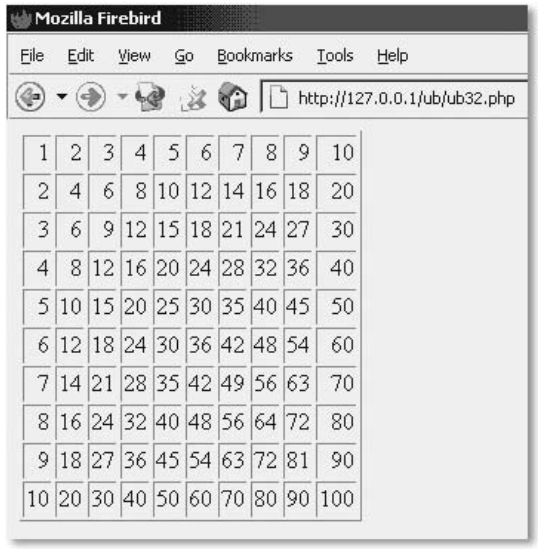


8/1	8/2	8/3	8/4	8/5
9/1	9/2	9/3	9/4	9/5
10/1	10/2	10/3	10/4	10/5
11/1	11/2	11/3	11/4	11/5
12/1	12/2	12/3	12/4	12/5
13/1	13/2	13/3	13/4	13/5
14/1	14/2	14/3	14/4	14/5
15/1	15/2	15/3	15/4	15/5

Abbildung B.38 Geschachtelte Schleife und Tabelle

Übung UB32

Erweitern Sie das Programm aus Übung `UB29`. Betten Sie das kleine Einmaleins in eine Tabelle ein (`ub32.php`). Die Ausgabe soll wie folgt aussehen:



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Abbildung B.39 »Kleines Einmaleins« in Tabelle

B.5.5 while-Schleife

while

Die `while`-Schleife wird dazu benutzt, eine unbestimmte Anzahl an Wiederholungen zu erzeugen. Das Ende der Wiederholungen wird bei einem der Schleifendurchläufe erreicht, es kann nicht vorher errechnet werden. `while`-Schleifen werden häufig bei Datenbankabfragen eingesetzt (siehe Abschnitt D.3, MySQL-Datenbanken).

Im nachfolgenden Beispiel wird gewürfelt. Die gewürfelten Zahlen werden addiert. Dies wird wiederholt, solange die Summe der gewürfelten Zahlen kleiner als `30` ist.

Zufallsgenerator

Zum Erzeugen der »zufälligen« Würfelergebnisse wird der Zufallsgenerator von PHP verwendet. Er muss zunächst initialisiert werden, damit er »tatsächlich zufällige« Ergebnisse produziert. Innerhalb der Schleife wird jeweils ein Würfelergebnis erzeugt. Die dazu notwendigen Funktionen `srand()` und `rand()` werden in Abschnitt F.4, Mathematische Funktionen, noch näher erläutert.

Die Anzahl der Würfe ist sowohl dem Entwickler als auch dem Benutzer unbekannt, daher kann keine `for`-Schleife verwendet werden. Das Programm sieht wie folgt aus:

```
<html>
<body>
<?php
   /* Initialisierung */
   srand((double)microtime()*1000000);
   $summe = 0;
   while ($summe < 30)
   {
      $zufallszahl = rand(1,6);       // Würfel
      $summe = $summe + $zufallszahl;
      echo "Zahl $zufallszahl, Summe $summe<br>";
   }
?>
</body>
</html>
```

Listing B.27 Datei ub33.php

Die Bedingung zur Wiederholung muss in Klammern stehen. Bei der ersten Prüfung der Bedingung hat `$summe` noch den Wert `0`, deshalb darf die Schleife durchlaufen werden. Innerhalb der Schleife wird die gewürfelte Zufallszahl zur Variablen `$summe` addiert. Die gewürfelte Zahl und die aktuelle Zwischensumme werden ausgegeben.

Es wird wiederum daraufhin überprüft, ob die Summe noch kleiner als `30` ist. Ist dies der Fall, so wird die Schleife erneut durchlaufen. Anderenfalls wird mit der Anweisung hinter dem Schleifenende fortgefahren. Falls dort keine Anweisung mehr steht, ist das Programm zu Ende. Es wird also so lange (engl.: `while`) eine Zahl addiert, bis die Bedingung für die Wiederholung nicht mehr gilt.

Die Seite hat zum Beispiel folgendes Aussehen, natürlich abhängig von den zufällig ermittelten Werten:

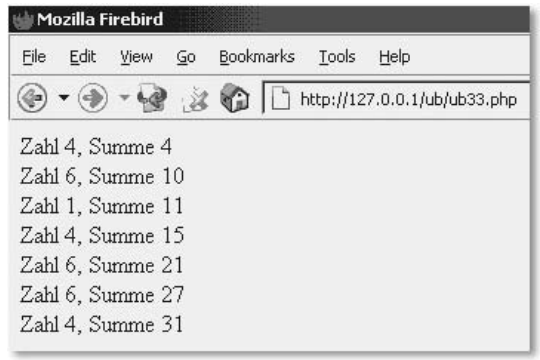


Abbildung B.40 while-Schleife

Übung UB34

Erstellen Sie ein kleines Computerspiel. Zwei Spieler würfeln gegeneinander (Zufallsgenerator), die Würfe jedes Spielers werden addiert. Sobald einer der beiden Spieler oder beide Spieler in einer Spielrunde die Zahl `30` erreicht oder überschritten haben, ist das Spiel zu Ende (Datei `ub34.php`). Der Name des Gewinners soll anschließend ausgegeben werden. Die Ausgabe könnte wie folgt aussehen:

Abbildung B.41 Spiel

B.5.6 Schleifenabbruch mit break

break

Mit Hilfe der Anweisung `break`, die bereits aus der `switch-case`-Verzweigung bekannt ist, kann eine Schleife vorzeitig beendet werden. Damit wird eine zusätzliche Möglichkeit für eine Schleifensteuerung geschaffen, die ein Programm unter Umständen besser lesbar machen kann.

Hinweis: Eine `break`-Anweisung innerhalb einer Schleife wird immer gemeinsam mit einer `if`-Bedingung auftreten, da der vorzeitige Abbruch einer Schleife nur in einem »Sonderfall« erfolgen sollte.

Im nachfolgenden Beispiel wird wiederum gewürfelt, solange die Summe kleiner als 30 ist. Es soll allerdings maximal neun Mal gewürfelt (Sonderfall) und dann abgebrochen werden.

```
<html>
<body>
<?php
   srand((double)microtime()*1000000);
   $summe = 0;
   $zaehler = 0;
   while ($summe < 30)
   {
      $zufallszahl = rand(1,6);
```

```
        $summe = $summe + $zufallszahl;
        $zaehler = $zaehler + 1;
        echo "Nr. $zaehler, Zahl $zufallszahl,";
        echo " Summe $summe<p>";
        if ($zaehler >= 9) break;       // Sonderfall
    }
?>
</body>
</html>
```

Listing B.28 Datei ub35.php

Es wird ein zusätzlicher Zähler verwendet (Variable `$zaehler`). Diese Variable wird zunächst auf `0` gesetzt. Innerhalb der Schleife wird ihr Wert immer um `1` erhöht. Sie zählt also die Anzahl der Schleifendurchläufe. Falls dabei die Zahl `9` erreicht beziehungsweise überschritten wird, bricht die Schleife unmittelbar ab. Dies geschieht auch dann, wenn die Summe noch kleiner als `30` ist.

Die Seite hat zum Beispiel folgendes Aussehen, natürlich abhängig von den zufällig ermittelten Werten:

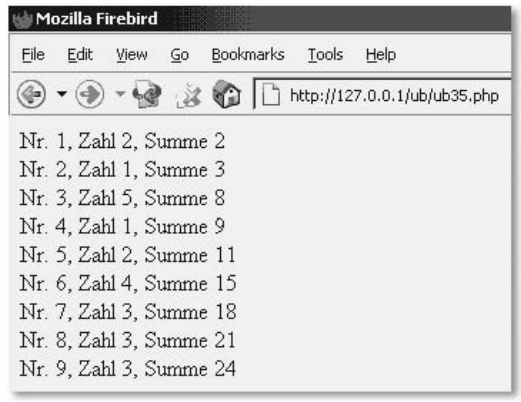


Abbildung B.42 Beispiel zu break

Hinweis: Der Vergleich `if ($zaehler == 9)` hätte auch zu einem Abbruch geführt, allerdings nur bei einer Erhöhung um `1`. Falls man zum Beispiel den Zähler immer um `2` erhöhen würde, könnte der Wert `9` nicht genau erreicht werden. Die Schleife würde über die vorgesehene Abbruchstelle hinausgehen. Daher arbeitet man an diesen Stellen normalerweise mit Bereichsangaben (>= oder <=).

B.5.7 Weitere Schleifenanweisungen

Im Zusammenhang mit Schleifen existieren in der Sprache PHP noch die nachfolgenden Anweisungen. Sie sind als Alternativen zu sehen und bieten für bestimmte Fälle Vorteile.

do-while

Die `do-while`-Schleife verhält sich wie eine `while`-Schleife, zusätzlich aber gilt: Die Schleife wird mindestens einmal durchlaufen, da die Bedingung für die Wiederholung erst am Ende geprüft wird. Die Syntax lautet: do-while

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

foreach

Die `foreach`-Schleife wird im Zusammenhang mit assoziativen Feldern verwendet. Sie ermöglicht die Bearbeitung aller Elemente eines solchen Feldes mit einer Schleife (siehe Abschnitt B.6, Felder). Die Syntax lautet: foreach

```
foreach([Feld-Ausdruck])
{
[Anweisungen]
}
```

continue

Die Anweisung `continue` wird verwendet, um aufgrund einer Bedingung den Rest einer Schleife zu überspringen und unmittelbar mit dem nächsten Schleifendurchlauf fortzusetzen. continue

B.6 Felder

Zur Speicherung einer größeren Menge von zusammengehörigen Daten kann man entweder viele einzelne Variablen, jeweils mit einem eigenen Namen, oder ein Feld von Variablen mit einem einheitlichen Namen nutzen. Felder sind dabei zu bevorzugen, da sie eine schnellere und komfortablere Verarbeitung bieten. PHP unterstützt zwei Typen von Feldern:

numerisch indizierte Felder

- numerisch indizierte Felder: Die einzelnen Variablen in einem numerisch indizierten Feld werden über eine laufende Nummer innerhalb des Feldes angesprochen,

assoziative Felder

- assoziative Felder (auch Hash-Tabelle genannt): Die einzelnen Variablen in einem assoziativen Feld werden über eine eindeutige Bezeichnung innerhalb des Feldes angesprochen.

Beide genannten Feldtypen werden in diesem Abschnitt angesprochen. Ein ausführlicher Abschnitt über Felder findet sich in Kapitel F, Weitere Themen.

Felder können eine oder mehrere Dimensionen haben.

Feldmodell

- Ein eindimensionales Feld kann man sich als einen mathematischen Vektor oder einfach als eine Art Liste vorstellen. Dies könnte zum Beispiel eine Preisliste oder die Namensliste von Mitgliedern einer Gruppe sein.
- Ein zweidimensionales Feld kann man sich als eine mathematische Matrix oder einfach als eine Tabelle vorstellen. Dies könnte zum Beispiel der Inhalt einer Datenbanktabelle mit verschiedenen Feldern und Datensätzen sein. Zweidimensionale Felder gibt es in drei Varianten: rein numerisch indiziert, rein assoziativ oder gemischt (numerisch indiziert/assoziativ).
- Es können auch Felder mit mehr als zwei Dimensionen zum Einsatz kommen. Eine geeignete Modellvorstellung wird mit wachsender Dimensionszahl allerdings immer schwerer.

B.6.1 Numerisch indizierte Felder

Nehmen wir an, es wäre eine Woche lang jeden Tag an einem bestimmten Ort eine Temperatur gemessen worden. Es stehen somit sieben Temperaturwerte zur weiteren Betrachtung und Untersuchung zur Verfügung. Diese Werte werden zunächst in einem numerisch indizierten Feld gespeichert und ausgegeben:

```
<html>
<body>
<?php
   $tp = array(17.5, 19.2, 21.8, 21.6, 17.5);
   $tp[5] = 20.2;
   $tp[6] = 16.6;
   for($i=0; $i<=6; $i = $i+1)
```

```
    {
        echo "$tp[$i] <br>";
    }
?>
</body>
</html>
```

Listing B.29 Datei ub36.php

In diesem Programm werden zwei häufig eingesetzte Techniken zur Erzeugung beziehungsweise Vergrößerung von Feldern gezeigt:

- Mit Hilfe der Funktion `array()` wird die Variable `$tp` zu einem Feld (engl.: array) mit fünf Elementen. Diese Elemente sind automatisch durchnummeriert worden, beginnend bei `0`. **array()**
- Felder können auch einfach durch die Zuweisung einzelner Elemente erzeugt oder vergrößert werden. Dies ist hier mit den beiden Zuweisungen `$tp[5] = 20.2;` und `$tp[6] = 16.6;` geschehen. Dabei ist die bisherige Nummerierung zu beachten, sonst könnten vorhandene Elemente überschrieben werden.
- Ein einzelnes Feld-Element wird angesprochen, indem man nach dem Namen des Feldes in eckigen Klammern die laufende Nummer des Elementes angibt. Diese wird auch Index genannt. **Feldindex**

Insgesamt hat das Feld nun sieben Elemente und die folgende Struktur:

Name des Elementes	Nummer (= Index) des Elementes	Wert des Elementes
`$tp[0]`	`0`	`17.5`
`$tp[1]`	`1`	`19.2`
`$tp[2]`	`2`	`21.8`
`$tp[3]`	`3`	`21.6`
`$tp[4]`	`4`	`17.5`
`$tp[5]`	`5`	`20.2`
`$tp[6]`	`6`	`16.6`

Diese Elemente werden anschließend mit Hilfe einer `for`-Schleife untereinander ausgegeben. Dabei nimmt die Schleifenvariable `$i` nacheinander die verwendeten Indexwerte an (`0` bis `6`).

Die Ausgabe sieht wie folgt aus:

Abbildung B.43 Numerisch indiziertes Feld

Übung UB37

Es sollen Vorname und Alter von sechs verschiedenen Personen in zwei Feldern gespeichert werden. Das erste Feld soll die Vornamen enthalten, das zweite Feld die zugehörigen Altersangaben. Die Elemente der beiden Felder sollen paarweise in der folgenden Form als Tabelle auf dem Bildschirm ausgegeben werden (Datei `ub37.php`):

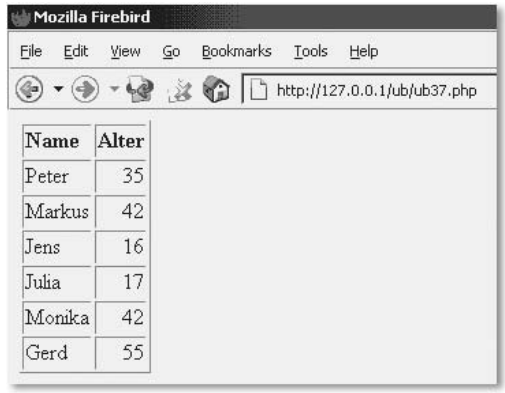


Abbildung B.44 Ergebnis Übung UB37

B.6.2 Assoziative Felder

Die Temperaturwerte aus dem vorherigen Abschnitt sollen nun in einem assoziativen Feld angeordnet werden. Die Elemente eines solchen Feldes werden nicht über eine laufende Nummer, sondern über eine Schlüsselbezeichnung (Key) identifiziert. Dadurch wird es möglich, den Feld-Elementen eindeutige Begriffe zuzuordnen und die Suche nach bestimmten Feld-Elementen zu vereinfachen.

Zunächst sollen die Werte wiederum gespeichert und ausgegeben werden:

```
<html>
<body>
<?php
   $tp = array("Montag"=>17.5,
       "Dienstag"=>19.2, "Mittwoch"=>21.8);
   $tp["Donnerstag"] = 21.6;
   $tp["Freitag"] = 17.5;
   $tp["Samstag"] = 20.2;
   $tp["Sonntag"] = 16.6;

   // Ein bestimmtes Element
   echo $tp["Montag"] . "<p>";

   // Tabellenkopf
   echo "<table border>";
   echo "<tr><td><b>Wochentag</b></td>";
   echo "<td><b>Temperatur</b></td></tr>";

   // Alle Keys und Values aus dem Feld
   foreach($tp as $name=>$wert)
   {
      echo "<tr> <td>$name</td>
        <td align='right'>$wert</td> </tr>";
   }
   echo "</table>";

   // Nur alle Values aus dem Feld, zum Summieren
   $summe = 0;
   foreach($tp as $wert)
   {
      $summe = $summe + $wert;
   }
   echo "<br>Summe: $summe";
?>
</body>
</html>
```

Listing B.30 Datei ub38.php

Die Verwendung von assoziativen Feldern erscheint zunächst etwas unübersichtlich. Nachdem man sich aber mit der Vorgehensweise vertraut gemacht hat, können assoziative Felder je nach Problemstellung einige Vorteile mit sich bringen.

Auch hier werden gleich zwei Techniken zur Erzeugung eines Feldes gezeigt:

Key, Value

- Mit Hilfe der Funktion `array()` wird die Variable `$tp` zu einem Feld mit drei Elementen. Diese Elemente haben eindeutige Schlüsselbezeichnungen (Keys) und zugehörige Werte (Values). Diese Paare werden einander mit dem Operator => zugeordnet. Der Key muss dabei zwischen doppelte Hochkommata geschrieben werden.
- Felder können auch einfach durch die Zuweisung einzelner Elemente erzeugt oder vergrößert werden. Dies ist hier mit den Zuweisungen in der Form `$tp["Samstag"] = 20.2;` usw. geschehen.

Insgesamt hat das Feld nun sieben Elemente und die folgende Struktur:

Name des Elementes	Schlüsselbezeichnung (Key) des Elementes	Wert (Value) des Elementes
`$tp["Montag"]`	`Montag`	`17.5`
`$tp["Dienstag"]`	`Dienstag`	`19.2`
`$tp["Mittwoch"]`	`Mittwoch`	`21.8`
`$tp["Donnerstag"]`	`Donnerstag`	`21.6`
`$tp["Freitag"]`	`Freitag`	`17.5`
`$tp["Samstag"]`	`Samstag`	`20.2`
`$tp["Sonntag"]`	`Sonntag`	`16.6`

Eine Möglichkeit, ein Element eines assoziativen Feldes auszugeben:

```
echo $tp["Montag"] . "<p>";
```

Hinweis: Da der Name des Keys zwischen doppelte Hochkommata geschrieben werden muss, ist die Ausgabe innerhalb einer Zeichenkette nicht möglich. Eine der folgenden Vorgehensweisen hätte also nicht zum Erfolg geführt:

- Es werden überall doppelte Hochkommata gesetzt, dadurch wird die Zeichenkette zu früh beendet. Beispiel: `echo "$tp["Montag"]<p>";`

- Es werden statt doppelter Hochkommata bei den Keys einfache Hochkommata gesetzt. Dies ist ein Fehler bei der Benutzung des assoziativen Feldes und führt dazu, dass der Index nicht erkannt wird. Beispiel: `echo "$tp['Montag']<p>";`
- Es werden bei der echo-Anweisung einfache Hochkommata gesetzt. Dies ist erlaubt, liefert aber nur den Namen des Feld-Elementes, nicht jedoch seinen Wert. Beispiel: `echo '$tp["Montag"]<p>';`

foreach, as

Die `foreach`-Schleife bietet eine Möglichkeit, alle Elemente eines assoziativen Feldes auszugeben:

- In der ersten Schleife sorgt die Anweisung `foreach($tp as $name=>$wert)` dafür, dass bei jedem Schleifendurchlauf jeweils ein einzelnes Key-Value-Paar in den Variablen `$name` und `$wert` bereitgestellt wird. Beide Variablen werden ausgegeben.
- In der zweiten Schleife sorgt die Anweisung `foreach($tp as $wert)` dafür, dass bei jedem Schleifendurchlauf jeweils nur der Value jedes Elementes in der Variablen `$wert` bereitgestellt wird. Dieser Wert wird zum Summieren aller Feld-Elemente genutzt.

Die Ausgabe sieht wie folgt aus:

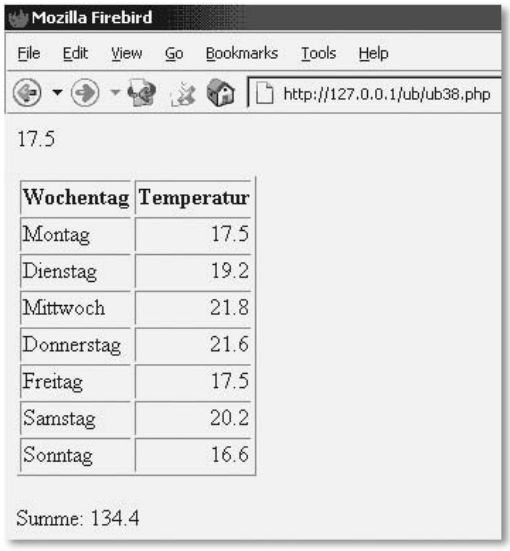


Abbildung B.45 Assoziatives Feld

Hinweis: Falls man einem bestimmten Key bei der Erzeugung des Feldes oder später einen neuen Wert zuordnet, so wird nicht etwa ein

neues Element hinzugefügt, sondern der erste Wert wird überschrieben. Die folgende Anweisung erzeugt also nur die beiden Feld-Elemente mit den Keys `Montag` und `Dienstag` und den Values `19.2` und `21.8`:

```
$tp = array("Montag"=>17.5, "Dienstag"=>19.2,
      "Montag"=>21.8);
```

Übung UB39

Es sollen Vorname und Alter von sechs verschiedenen Personen untersucht werden. Diese sechs Angaben werden in einem assoziativen Feld gespeichert. Die Vornamen sollen die Keys, die Altersangaben die Values darstellen. Key und Value der Elemente des Feldes sollen paarweise in der folgenden Form als Tabelle auf dem Bildschirm ausgegeben werden (Datei `ub39.php`):

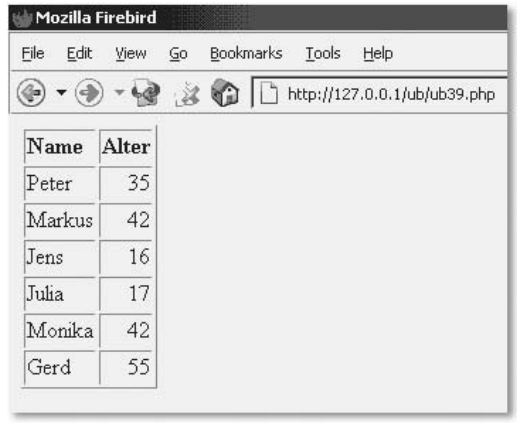


Abbildung B.46 Ergebnis Übung UB39

B.7 Funktionen

Es gibt in PHP zahlreiche vordefinierte Funktionen, die vom Entwickler eingesetzt werden können. Diese werden in einem eigenen Kapitel beschrieben. Darüber hinaus hat der Entwickler die Möglichkeit, eigene Funktionen zu schreiben, so genannte benutzerdefinierte Funktionen. Diese bringen folgende Vorteile:

- Gleiche oder ähnliche Vorgänge müssen nur einmal beschrieben werden und können beliebig oft ausgeführt werden.

- Programme werden modularisiert, das heißt, sie werden in kleinere Bestandteile zerlegt, die übersichtlicher sind und einfacher gewartet werden können. **Modularisierung**

Eine besondere Variante von Funktionen, die so genannten rekursiven Funktionen, werden in einem anschaulichen Beispiel im Abschnitt F.2.11, Informationen über Verzeichnisbaum, erläutert.

B.7.1 Ein erstes Beispiel

Ein Beispiel für eine einfache, benutzerdefinierte Funktion:

```
<html>
<head>
<?php
   function trennstrich()
   {
      echo "<br>";
      for ($i=1; $i<=40; $i=$i+1)
         echo "-";
      echo "<br>";
   }
?>
</head>
<body>
<?php
   trennstrich();
   echo "Dies ist ein Programm,";
   trennstrich();
   echo "in dem mehrmals";
   trennstrich();
   echo "eine Funktion verwendet wird,";
   trennstrich();
   echo "die zu Beginn definiert wurde";
   trennstrich();
?>
</body>
</html>
```

Listing B.31 Datei ub40.php

Eigene Funktionen werden mit Hilfe von `function ... () { ... }` definiert. Der Name der Funktion folgt nach dem Schlüsselwort **function ()**

`function`, in runden Klammern folgen die Parameter, falls vorhanden. Anschließend folgt in geschweiften Klammern der eigentliche Funktionsrumpf. Meist erfolgt die Definition einer Funktion im Kopf eines HTML-Dokumentes, wie hier bei der Funktion `trennstrich()`.

Aufgabe der Funktion `trennstrich()` ist die Darstellung eines Zeilenumbruchs, von `40` Bindestrichen und wiederum eines Zeilenumbruchs. Jedes Mal, wenn sie vom eigentlichen Programm im Rumpf des Dokumentes aufgerufen wird (mit `trennstrich()`), führt sie die genannte Aufgabe aus.

Die Seite hat das folgende Aussehen:

Abbildung B.47 Funktion trennstrich()

Übung UB41

Erstellen Sie eine Funktion `vermerk()`, die einen Entwicklervermerk erzeugt: Jedes Mal, wenn die Funktion aufgerufen wird, erscheint Ihr Name in einer Tabellenzelle mit Rahmen, wie nachfolgend dargestellt. Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals aufgerufen wird (Datei `ub41.php`). Die Ausgabe könnte wie folgt aussehen:

Abbildung B.48 Ergebnis Übung UB41

B.7.2 Definition, Aufruf, Funktionstypen

Falls der Entwickler bestimmte nützliche Funktionen geschrieben hat, die er in mehreren Programmen verwenden möchte, so können diese Funktionen auch in externen Dateien untergebracht werden. Diese externen Dateien können mit den Anweisungen `require` beziehungsweise `include` in die jeweiligen Programme eingebunden werden.

Der Aufruf einer eigenen oder einer vordefinierten Funktion erfolgt — **Aufruf**

- entweder aus dem Rumpf des Dokumentes heraus, im oben angegebenen Beispiel mit `trennstrich()`, oder
- aus anderen Funktionen heraus.

Dabei ist der Ort der Funktionsdefinition wichtig. Man kann nur Funktionen aufrufen, die dem Programm bekannt sind. Sie müssen also — **Definition**

- entweder zu den vordefinierten Funktionen gehören oder
- im Dokument definiert worden sein, wie im oben angegebenen Beispiel, oder
- aus eigenen, externen Dateien stammen, die mit `require` beziehungsweise `include` bereits eingebunden worden sind. — **require, include**

Man unterscheidet zwischen folgenden Funktionstypen:

- Funktionen ohne Parameter: Diese Funktionen führen bei jedem Aufruf immer genau die gleiche Aufgabe aus, wie im oben angegebenen Beispiel.
- Funktionen mit einem oder mehreren Parametern: Diese Funktionen führen bei jedem Aufruf in Abhängigkeit von den Parametern ähnliche Aufgaben aus.

- Funktionen mit Rückgabewerten: Diese Funktionen führen gleiche oder ähnliche Aufgaben aus und liefern ein Ergebnis an die aufrufende Stelle zurück.

Für den Namen einer Funktion gelten die gleichen Regeln wie für den Namen einer Variablen. Der einzige Unterschied besteht darin, dass Namen von Funktionen nicht mit dem Zeichen $ (Dollar) beginnen dürfen. Die Regeln wurden bereits in Abschnitt B.2.1, Namen, erwähnt (siehe dort).

B.7.3 Funktionen mit einem Parameter

Parameter

Eine Funktion mit einem Parameter führt bei jedem Aufruf in Abhängigkeit vom Parameterwert ähnliche Aufgaben aus. Das vorherige Beispiel wurde etwas erweitert, die Funktion erzeugt nun unterschiedlich lange Trennstriche, wie nachfolgend zu erkennen ist:

```
<html>
<head>
<?php
   function trennstrich($anzahl)
   {
      echo "<br>";
      for ($i=1; $i<=$anzahl; $i=$i+1)
         echo "-";
      echo "<br>";
   }
?>
</head>
<body>
<?php
   trennstrich(30);
   echo "In diesem Programm,";
   trennstrich(40);
   echo "sind die Trennstriche";
   $x = 20;
   trennstrich($x);
   echo "unterschiedlich lang";
   trennstrich($x * 3);
?>
```

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

Listing B.32 Datei ub42.php

Die Funktion `trennstrich()` wird insgesamt vier Mal aufgerufen, jedes Mal mit einem anderen Wert in den Klammern hinter dem Funktionsnamen. Dies ist der Parameter; er kann eine Zahl, eine Variable oder das Ergebnis einer Berechnung sein.

Er wird an die Funktion übergeben, dort wird dieser Wert in der Variablen `$anzahl` gespeichert. Der Wert von `$anzahl` steuert die Ausführung der `for`-Schleife mit dem Ergebnis, dass die Trennstriche unterschiedlich lang sind. Es wird also bei jedem Aufruf fast die gleiche Aktion durchgeführt, beeinflusst durch den Wert des Parameters.

Die Ausgabe sieht wie folgt aus:

Abbildung B.49 Ergebnis Funktion trennstrich() mit Parameter

Übung UB43

Erweitern Sie die Funktion `vermerk()` aus Übung `UB41`. Sie soll von verschiedenen Entwicklern genutzt werden können. Der Name des Entwicklers wird als Parameter an die Funktion übergeben. Jedes Mal, wenn die Funktion aufgerufen wird, erscheint der betreffende Name in einer Tabellenzelle mit Rahmen und fester Größe, wie nachfolgend dargestellt. Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals mit verschiedenen Namen aufgerufen wird (Datei `ub43.php`).

Abbildung B.50 Ergebnis Übung UB43

Übung UB44

Erstellen Sie eine Funktion `quadrat()`, die das Quadrat einer Zahl berechnet und ausgibt. Die betreffende Zahl wird als Parameter an die Funktion übergeben. Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals mit verschiedenen Zahlen aufgerufen wird (Datei `ub44.php`). Nachfolgend ein Beispiel:

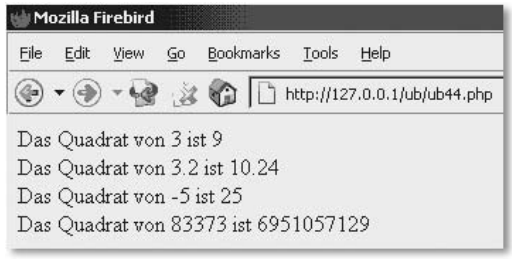


Abbildung B.51 Ergebnis Übung UB44

B.7.4 Funktionen mit mehreren Parametern

mehrere Parameter

Falls einer Funktion mehrere Parameter übergeben werden, sind Anzahl und Reihenfolge der Parameter wichtig. Der erste Wert wird an den ersten Parameter, der zweite Wert an den zweiten Parameter übergeben usw. Ein Beispiel für eine eigene Funktion mit mehreren Parametern:

```
<html>
<head>
<?php
   function flexloop($von, $bis, $schritt)
   {
      echo "<br>Es folgt eine Schleife von $von";
```

```
      echo " bis $bis mit der S.-Weite $schritt<br>";

      for ($i = $von; $i <= $bis; $i = $i + $schritt)
      {
         echo "$i ";
      }
   }
?>
</head>
<body>
<?php
   echo "<p>Nummer 1";
   flexloop(5,27,3);

   echo "<p>Nummer 2";
   flexloop(-10,10,4);

   echo "<p>Nummer 3";
   $x = 100;
   $y = 200;
   $z = 10;
   flexloop($x,$y,$z);

   echo "<p>Nummer 4";
   flexloop($x,$y,($y-$x)/8);
?>
</body>
</html>
```

Listing B.33 Datei ub45.php

Beim Aufruf der Funktion `flexloop()` müssen jeweils drei Parameter übergeben werden, und zwar durch Kommata voneinander getrennt. Diese werden in der vorliegenden Reihenfolge den Variablen `$von`, `$bis` und `$schritt` zugeordnet. Diese Variablen werden zur Steuerung der `for`-Schleife in der Funktion verwendet. Es wird also bei jedem Aufruf eine ähnliche Aktion durchgeführt, beeinflusst durch den Wert der Parameter. Die Ausgabe sieht folgendermaßen aus:

Abbildung B.52 Funktion mit mehreren Parametern

Übung UB46

Schreiben Sie ein Programm (Datei `ub46.php`), in dem eine Funktion `mittel()` definiert und benutzt wird, die den arithmetischen Mittelwert von drei Zahlen berechnet und ausgibt. Die drei Zahlen werden der Funktion als Parameter übergeben. Testen Sie die Funktion mit mehreren verschiedenen Aufrufen innerhalb des Programms. Hinweis: Der arithmetische Mittelwert von drei Zahlen wird berechnet, indem man die Summe der drei Zahlen durch drei teilt. Die Ausgabe könnte wie folgt aussehen:

Abbildung B.53 Ergebnis Übung UB46

Übung UB47

Erweitern Sie die Funktion `vermerk()` aus Übung `UB43`. Sie soll von verschiedenen Entwicklern genutzt werden können. Vorname, Nachname und Abteilung werden als Parameter an die Funktion übergeben. Jedes Mal, wenn die Funktion aufgerufen wird, erscheint eine Ausgabezeile mit diesen Informationen und der E-Mail-Adresse. Die E-Mail-Adresse setzt sich wie folgt zusammen: `vorname.nachname@abteilung.phpdevel.de`. Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals mit verschiedenen Informationen aufgerufen wird (Datei `ub47.php`). Eine mögliche Ausgabe:

Abbildung B.54 Ergebnis Übung UB47

B.7.5 Rückgabewert einer Funktion

Rückgabewert

Funktionen mit Rückgabewert dienen dazu, ein Ergebnis zu ermitteln und dieses an die aufrufende Stelle zurückzuliefern. Der zurückgelieferte Wert muss entweder in einer Variablen gespeichert oder direkt ausgegeben werden, anderenfalls geht er verloren. Ein Beispiel für eine Funktion mit Rückgabewert:

```
<html>
<head>
<?php
   function add($z1, $z2)
   {
      $summe = $z1 + $z2;
      return $summe;
   }
?>
</head>
```

```
<body>
<?php
   $c = add(3,4);      // aufrufende Stelle
   echo "Summe: $c<br>";

   $x = 5;
   $c= add($x,12);     // aufrufende Stelle
   echo "Summe: $c<br>";

   // aufrufende Stelle innerhalb der Ausgabe
   echo "Summe: " . add(13,2) . "<br>";

   // Ausgabe in Zeichenkette, falsch!
   echo "Summe: add(13,2)<br>";
?>
</body>
</html>
```

Listing B.34 Datei ub48.php

Die Funktion `add()` hat die beiden Parameter `$z1` und `$z2`. Innerhalb der Funktion werden diese beiden Parameter addiert und in der Variablen `$summe` gespeichert.

return Mit Hilfe der Anweisung `return` wird dieser Wert an die aufrufende Stelle zurückgeliefert und kann dort weiter verarbeitet werden. In den ersten beiden Fällen wird der Wert in der Variablen `$c` gespeichert, im dritten Fall ohne Zwischenspeicherung direkt ausgegeben.

Die Ausgabe sieht wie folgt aus:

Abbildung B.55 Funktion mit Rückgabewert

Hinweis: Eine solche direkte Ausgabe eines Funktionsergebnisses darf nicht innerhalb einer Zeichenkette stehen. Die letzte Zeile der Ausgabe

zeigt, dass dann nur der Name der Funktion und ihre Parameter genannt werden, aber die Funktion nicht aufgerufen wird.

Hinweis: Mit Hilfe der Anweisung `return` kann eine Funktion auch vorzeitig verlassen werden. Dies ist unabhängig davon, ob sie einen Wert zurückliefert oder nicht.

Hinweis: Mit Hilfe der Anweisung `return` kann nicht nur eine Einzelvariable, sondern auch ein Feld aus einer Funktion zurückgeliefert werden. Ein Beispiel: Der Aufruf `$x = feldfunc();` sorgt dafür, dass `$x` zu einem Feld mit mehreren Elementen wird. Dies setzt voraus, dass in der Funktion `feldfunc()` eine Anweisung wie zum Beispiel `return $p` existiert und `$p` ein (numerisch indiziertes oder assoziatives) Feld mit mehreren Elementen ist.

Übung UB49

Schreiben Sie ein Programm (Datei `ub49.php`), in dem eine Funktion `bigger()` definiert und aufgerufen wird. Diese Funktion ermittelt die größere von zwei übergebenen Zahlen und liefert diese Zahl zurück. Testen Sie die Funktion mit mehreren verschiedenen Aufrufen innerhalb des Programms und geben Sie das Ergebnis zur Kontrolle aus.

Ein Aufruf der Funktion könnte lauten:

```
$c = bigger(3,4);
```

Die Ausgabe des Programms wäre in diesem Falle:

```
Maximum: 4
```

B.7.6 Kopie und Referenz

Bei der Übergabe von Parametern an eine Funktion muss man sich noch die Frage stellen: Was passiert, falls ich in der Funktion einen der Parameter verändere ?

PHP bietet hier mehrere Möglichkeiten an:

- Übergabe der Parameter als Kopie (call-by-value), eine Veränderung der Kopien hat keine Rückwirkung auf das Original: Diese Methode wird angewendet, falls man der Funktion nur Werte übergeben möchte. Sie wurde bei den bisherigen Programmen für Funktionen angewendet. **call-by-value**
- Übergabe der Parameter als Referenz auf das Original (call-by-reference), eine Veränderung hat Rückwirkung auf das Original: Diese **call-by-reference**

Methode wird angewendet, wenn mehr als ein Wert aus einer Funktion zurückgeliefert werden soll. Über einen Rückgabewert (siehe Kapitel B.7.5, Rückgabewert einer Funktion) könnte nur ein einziger Wert zurückgeliefert werden.

- Übergabe von Referenzen auf die Original-Parameter (ebenfalls: call-by-reference), eine Veränderung hat Rückwirkung auf das Original: Möchte man die Änderungsmöglichkeiten der zweiten Methode mit der Sicherheit der ersten Methode verbinden, so kann man diese Vorgehensweise anwenden. Dabei kann der Entwickler von Fall zu Fall entscheiden, ob er beim Aufruf einer Funktion den Wert oder eine Referenz übergibt.

Alle drei Methoden sollen zum Vergleich an einem Beispiel dargestellt werden. Der Funktion `rtauschen()` beziehungsweise `vtauschen()` werden zwei Parameter übergeben. Innerhalb der Funktionen sollen die beiden übergebenen Parameter miteinander vertauscht werden. Abhängig von den verschiedenen angewendeten Methoden wird dieses Tauschen Rückwirkungen auf die Original-Variablen im Hauptprogramm haben. Die Werte werden jeweils vor und nach dem Tauschen angezeigt.

```
<html>
<head>
<?php
   function vtauschen($a, $b)
   {
      $temp = $a;
      $a = $b;
      $b = $temp;
   }
   function rtauschen(&$a, &$b)
   {
      $temp = $a;
      $a = $b;
      $b = $temp;
   }
?>
</head>
<body>
<?php
   $x = 12;   $y = 18;
```

```
   echo "Methode 1, vorher: $x, $y <br>";
   vtauschen($x,$y);
   echo "Methode 1, nachher: $x, $y <p>";

   $x = 12;   $y = 18;
   echo "Methode 2, vorher: $x, $y <br>";
   rtauschen($x,$y);
   echo "Methode 2, nachher: $x, $y <p>";

   $x = 12;   $y = 18;
   echo "Methode 3, vorher: $x, $y <br>";
   vtauschen(&$x,&$y);
   echo "Methode 3, nachher: $x, $y <p>";
?>
</body>
</html>
```

Listing B.35 Datei ub50.php

Methode 1: Der Wert der Variablen `$x` wird bei Aufruf der Funktion `vtauschen()` an die Variable `$a` übergeben. Der Wert der Variablen `$y` wird an die Variable `$b` übergeben. Innerhalb der Funktion `vtauschen()` werden `$a` und `$b` getauscht. Da aber nur die Kopien getauscht wurden, hat dies auf die Originale `$x` und `$y` keine Auswirkungen.

Referenz &

Methode 2: Den Unterschied sieht man im Funktionskopf `function rtauschen(&$a, &$b)`. Die Variable `$x` wird bei Aufruf der Funktion `rtauschen()` an eine Referenz (Zeichen `&`) übergeben, dies ist `$a`. Die Variable `$y` wird ebenfalls an eine Referenz übergeben, dies ist `$b`. Innerhalb der Funktion werden die Referenzen vertauscht. Dadurch werden auch die Originale `$x` und `$y` vertauscht.

Methode 3: Den Unterschied sieht man beim Aufruf der Funktion: `vtauschen(&$x, &$y);` Beim Aufruf der Funktion `vtauschen()` wird eine Referenz auf die Variable `$x` an die Variable `$a` übergeben. Außerdem wird eine Referenz auf die Variable `$y` beim Aufruf an die Variable `$b` übergeben. Innerhalb der Funktion werden `$a` und `$b` vertauscht. Dadurch werden auch die Originale `$x` und `$y` vertauscht.

Die Ausgabe der Seite, jeweils mit den Werten vor und nach der (erfolgreichen) Vertauschung:

Abbildung B.56 Kopie und Referenz

Felder übergeben

Das nachfolgende Programm zeigt, dass bei Feldern die gleichen Möglichkeiten zur Verfügung stehen. Wird also nur das Original des Feldes an eine Kopie übergeben (Methode 1), so wird durch eine Veränderung der Kopie des Feldes das Original-Feld nicht verändert. Bei einer Übergabe per Referenz (Methode 2 und 3) zeigt sich das gleiche Verhalten wie oben beschrieben.

```
<html>
<head>
<?php
   function vtauschen($f)
   {
      $temp = $f[0];
      $f[0] = $f[1];
      $f[1] = $temp;
   }
   function rtauschen(&$f)
   {
      $temp = $f[0];
      $f[0] = $f[1];
      $f[1] = $temp;
   }
?>
</head>
<body>
<?php
   $f[0] = 12;   $f[1] = 18;
   echo "Methode 1, vorher: $f[0], $f[1] <br>";
```

```
   vtauschen($f);
   echo "Methode 1, nachher: $f[0], $f[1] <p>";

   $f[0] = 12;   $f[1] = 18;
   echo "Methode 2, vorher: $f[0], $f[1] <br>";
   rtauschen($f);
   echo "Methode 2, nachher: $f[0], $f[1] <p>";

   $f[0] = 12;   $f[1] = 18;
   echo "Methode 3, vorher: $f[0], $f[1] <br>";
   vtauschen(&$f);
   echo "Methode 3, nachher: $f[0], $f[1] <p>";
?>
</body>
</html>
```

Listing B.36 Datei ub51.php

Übung UB52

Schreiben sie ein PHP-Programm (Datei `ub52.php`) mit einer Funktion `rechne()`. Dieser Funktion werden zwei Zahlen übergeben. Sie soll zwei Ergebnisse über die Parameterliste zurückliefern, zum einen die Summe der beiden übergebenen Zahlen, zum anderen das Produkt der beiden übergebenen Zahlen.

Alle beteiligten Zahlen sollen im Hauptteil des Programms, also außerhalb der Funktion, ausgegeben werden. Verwenden Sie zur Übergabe die dritte Methode (Übergabe von Verweisen auf die Original-Parameter).

Nach einem Funktionsaufruf mit den Parametern `5` und `7` und der anschließenden Ausgabe erscheint Folgendes:

Abbildung B.57 Ergebnis Übung UB52

B.7.7 Gültigkeitsbereich von Variablen

Variablen werden auch nach ihrem Gültigkeitsbereich unterschieden. Dies ist der Bereich, in dem die betreffende Variable mit ihrem Wert bekannt ist. Es gibt

lokal
- lokale Variablen: Diese werden innerhalb einer Funktion definiert und stehen nur innerhalb dieser Funktion zur Verfügung,

global
- globale Variablen: Diese werden außerhalb von Funktionen definiert und stehen nur außerhalb derselben zur Verfügung. Dies ist ein Unterschied zu vielen anderen Programmiersprachen,

superglobal
- superglobale Variablen: Bei diesen handelt es sich um PHP-Systemvariablen. Sie stehen sowohl innerhalb als auch außerhalb von Funktionen zur Verfügung. Zu diesen zählt das assoziative Feld `$_POST`, das die Namen und Werte von Formularfeldern zur Verfügung stellt.

Einige Regeln im Zusammenhang mit dem Gültigkeitsbereich von Variablen:

- Die Benutzung lokaler Variablen bietet den Vorteil, dass Variablen nicht so leicht aus Versehen an weit voneinander entfernten Stellen verändert werden können.
- Ein Parameter, der als Kopie an eine Funktion übergeben wird, ist dort lokal.
- Lokale Variablen gleichen Namens in unterschiedlichen Funktionen oder globale Variablen gleichen Namens haben nichts miteinander zu tun.
- Falls man eine globale Variable innerhalb einer Funktion benutzen möchte, so muss sie dort entweder mit dem Schlüsselwort `global` bekannt gemacht oder als Parameter übergeben werden.

Hinweis: Die Variablen der Funktionen eines Programms sollten immer »so lokal wie möglich« sein:

- Zum einen wird die Modularisierung des Programms verbessert, das heißt die Zerlegung eines Programms in übersichtliche Programmteile mit klar definierten Schnittstellen zwischen den Teilen.
- Zum anderen wird die Wiederverwendbarkeit der Funktionen für andere Programme erleichtert.

Schlüsselwort global

Ein Beispiel mit lokalen und globalen Variablen und dem Schlüsselwort `global`:

```
<html>
<head>
<?php
   function summiere()
   {
      echo "Variable z: $z<p>";
      global $x;
      $y = 35;
      $z = $x + $y;
      echo "Variable z: $z<p>";
   }
?>
</head>
<body>
<?php
      $x = 6;
      $y = 52;
      $z = $x + $y;
      summiere();
      echo "Variable z: $z<p>";
?>
</body>
</html>
```

Listing B.37 Datei ub53.php

In diesem Programm existieren insgesamt fünf unterschiedliche Variablen:

- Die beiden Variablen `$y` und `$z` in der Funktion `summiere()` sind nur dort lokal bekannt.
- Zum Zeitpunkt des ersten Ausgabebefehles in der Funktion existiert `$z` noch nicht. Daher kann für `$z` kein Wert ausgegeben werden.
- Anschließend bekommen `$y` und `$z` innerhalb der Funktionen einen Wert, `$z` kann nun ausgegeben werden.
- Nach Verlassen der Funktion `summiere()` sind beide Werte nicht mehr verfügbar.
- Im Hauptprogramm gibt es insgesamt drei Variablen: `$x`, `$y` und `$z`. Das Schlüsselwort `global` sorgt dafür, dass `$x` auch in der Funktion `summiere()` mit seinem Wert bekannt ist.

- `$y` und `$z` sind nur außerhalb von Funktionen bekannt. Sie haben hier auch andere Werte als zum Beispiel in der Funktion `summiere()`.

Die Ausgabe des Programms sieht wie folgt aus:

Abbildung B.58 Gültigkeitsbereich von Variablen

Es folgt ein Beispiel zur Gegenüberstellung von globalen und superglobalen Variablen aus einem Formular. Zunächst der Programmcode des Formulars:

```
<html>
<body>
Bitte tragen Sie Ihren Vornamen und Ihren Nachnamen ein.<
br>
Senden Sie anschließend das Formular ab.<p>
<form action = "ub54.php" method = "post">
    <input name = "vor"> Vorname<p>
    <input name = "nach"> Nachname<p>
    <input type = "submit">
    <input type = "reset">
</form>
</body>
</html>
```

Listing B.38 Datei ub54.htm

Der PHP-Code des Auswertungsprogramms, das beide Auswertungsmethoden nutzt:

```
<html>
<body>
<?php
   function ausgabe()
   {
```

```
        global $vor;
        echo "Guten Tag, $vor $nach<br>";
        echo "Guten Tag, " . $_POST["vor"] . " " .
              $_POST["nach"] . "<br>";
    }

    echo "Guten Tag, $vor $nach<br>";
    echo "Guten Tag, " . $_POST["vor"] . " " .
         $_POST["nach"] . "<br>";
    ausgabe();
?>
</body>
</html>
```

Listing B.39 Datei ub54.php

Die Variablen `$vor` und `$nach` sind global, daher im Hauptprogramm bekannt, und können ausgegeben werden. Innerhalb der Funktion `ausgabe()` sind sie nur bekannt, falls sie dort mit dem Schlüsselwort `global` bekannt gemacht werden. Da `$nach` nicht bekannt gemacht wurde, erfolgt eine Meldung statt einer Ausgabe dieser Variablen.

Die Elemente `vor` und `nach` des superglobalen Feldes `$_POST` sind im Hauptprogramm und in der Funktion `ausgabe()` bekannt und können an beiden Stellen ausgegeben werden.

Nach folgender Eingabe:

Abbildung B.59 Eingabeformular

sieht die Ausgabe wie folgt aus:

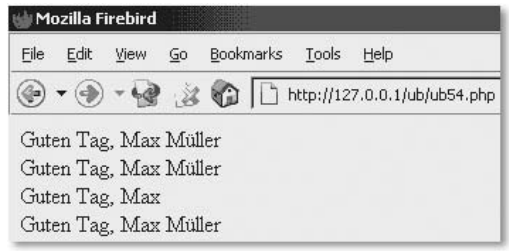


Abbildung B.60 Globale und superglobale Variablen

B.7.8 Variable Parameterlisten

Der Einsatz von Funktionen mit variablen Parameterlisten erhöht die Flexibilität von Funktionen, allerdings auch den Programmieraufwand.

variable Parameter-Anzahl

Bisher musste die Anzahl der Parameter bei einem Funktionsaufruf genau der Anzahl der Parameter entsprechen, die bei der Definition der Funktion vorgegeben wurden. Mit Hilfe der folgenden Funktionen ist dies nicht mehr zwingend notwendig:

func_num_args()

- Die Funktion `func_num_args()` liefert die Anzahl der übergebenen Parameter.

func_get_arg()

- Die Funktion `func_get_arg()` liefert einen bestimmten Parameter aus der Parameterliste.

func_get_args()

- Die Funktion `func_get_args()` (mit `s` am Ende) liefert ein numerisch indiziertes Feld mit allen übergebenen Parametern.

Das nachfolgende Programm verdeutlicht den Einsatz von `func_num_args()` und `func_get_arg()`.

```
<html>
<body>
<?php
   function addiere()
   {
      $anz = func_num_args();
      echo "Anzahl der Werte: $anz<br>";
      $sum = 0;
      for($i=0; $i<$anz; $i++)
      {
         $sum = $sum + func_get_arg($i);
      }
      echo "Summe der Werte: $sum<p>";
```

```
   }
   addiere(2,3,6);
   addiere(13,26);
   addiere(65,-3,88,31,12.5,7);
?>
</body>
</html>
```

Listing B.40 Datei ub55.php

Die Funktion `addiere()` wird insgesamt dreimal aufgerufen, jedes Mal mit einer anderen Anzahl an Parametern. Mit Hilfe von `func_num_args()` wird diese Anzahl ermittelt. Sie wird zur Steuerung einer `for`-Schleife verwendet. Innerhalb der `for`-Schleife werden alle gelieferten Parameter mit Hilfe von `func_get_arg()` ermittelt und anschließend addiert. Nach Beendigung der Schleife wird die Summe der Werte ausgegeben, siehe Darstellung:

Abbildung B.61 Variable Parameterlisten mit func_get_arg()

Eine alternative Lösung mit der Funktion `func_get_args()` bietet das folgende Programm:

```
<html>
<body>
<?php
   function addiere()
   {
      $param = func_get_args();
      $anz = func_num_args();
      echo "Anzahl der Werte: $anz<br>";
      $sum = 0;
```

```
        for($i=0; $i<$anz; $i++)
        {
           $sum = $sum + $param[$i];
        }
        echo "Summe der Werte: $sum<p>";
    }
    addiere(2,3,6);
    addiere(13,26);
    addiere(65,-3,88,31,12.5,7);
?>
</body>
</html>
```

Listing B.41 Datei ub56.php

Mit Hilfe der Anweisung `$param = func_get_args();` werden alle Parameter im Feld `$param` gespeichert. Die Funktion `func_num_args()` ermittelt wiederum die Anzahl der Parameter. Innerhalb der `for`-Schleife werden alle gelieferten Parameter aus dem Feld `$param` abgerufen und anschließend addiert.

Eine ähnliche Möglichkeit wird durch Parameter mit Voreinstellung (Default-Werte) geboten. Dazu Näheres in Abschnitt E.6, Optionale Parameter.

B.7.9 include-Anweisung

externe Funktionsbibliotheken

Benutzerdefinierte Funktionen, die von mehreren Programmen benutzt werden sollen, können in externe Dateien ausgelagert werden. Mit Hilfe der `include`-Anweisung wird der Inhalt dieser Dateien in das Programm eingebunden, welches sie benötigt. Dabei ist zu beachten, dass der Programmcode in den externen Dateien in gültige PHP-Markierungen eingeschlossen sein muss.

.inc.pcp

Hinweis: Es ist zu empfehlen, einer solchen Datei die Endung `.inc.php` zu geben. Zum einen ist sie damit als eine Datei erkennbar, die externe Funktionen beinhaltet. Zum anderen wird sie besser vor einem unberechtigten Aufruf geschützt:

- Abhängig von den Einstellungen des Webservers werden Dateien mit bestimmten Endungen einfach auf dem Bildschirm ausgegeben, so dass der Quellcode eingesehen werden kann.

- Dateien mit der Endung PHP werden vom Webserver als PHP-Code angesehen, intern gelesen und ausgeführt, so dass das Problem nicht entsteht.

Im nachfolgenden Beispiel wird zunächst innerhalb der Datei `ub57math.inc.php` eine Funktion `maxi()` definiert. Diese ermittelt aus den beiden übergebenen Parametern das Maximum, speichert ihn in die Variable `$erg` und liefert diesen Wert mit Hilfe der `return`-Anweisung zurück. Die `return`-Anweisung steht im vorliegenden Fall innerhalb des `if`-Blocks beziehungsweise innerhalb des `else`-Blocks.

```
<?php
   function maxi($x, $y)
   {
      if ($x > $y)
      {
         $erg = $x;
         return $erg;
      }
      else
      {
         $erg = $y;
         return $erg;
      }
   }
?>
```

Listing B.42 Datei ub57math.inc.php

Die Funktion wird vom nachfolgenden Programm aufgerufen. Dort wird zunächst die Datei `ub57math.inc.php` mit Hilfe der `include`-Anweisung eingebunden. Damit sind alle Funktionen aus der Datei `ub57math.inc.php` im aktuellen Programm bekannt und können verwendet werden.

```
<html>
<body>
<?php
   include "ub57math.inc.php";
   $a = 2;
   $b = 6;
   $c = maxi($a, $b);
   echo "Das Maximum von $a und $b ist $c";
```

```
?>
</body>
</html>
```

Listing B.43 Datei ub57.php

Die Ausgabe des Programms:

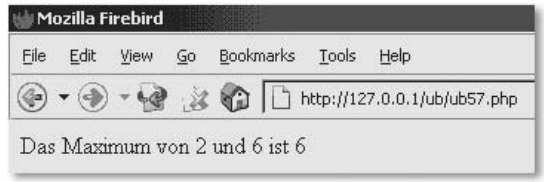


Abbildung B.62 Nutzung einer include-Datei

Übung UB58

Erstellen Sie eine kleine Funktionsbibliothek mit zwei Funktionen (Datei `ub58stat.inc.php`). Beide Funktionen sollen mit variablen Parameterlisten arbeiten.

- Die erste Funktion mit dem Namen `mittelwert()` soll den arithmetischen Mittelwert einer beliebigen Menge von Zahlen berechnen und per Rückgabewert zurückliefern können. Es muss also die Summe dieser Zahlen durch die Anzahl geteilt werden.
- Die zweite Funktion mit dem Namen `maximum()` soll die größte Zahl aus einer beliebigen Menge von Zahlen berechnen und per Rückgabewert zurückliefern können. Dazu ist folgende Vorgehensweise notwendig: Zunächst wird die erste übergebene Zahl einer lokalen Variablen (zum Beispiel `$mx`) der Funktion zugewiesen. Anschließend werden alle anderen übergebenen Zahlen mit `$mx` verglichen. Sollte eine der Zahlen größer als `$mx` sein, so hat man ein neues Maximum gefunden und dieser Wert wird `$mx` zugewiesen. Am Ende der Funktion wird `$mx` zurückgeliefert.

Testen Sie Ihre Bibliothek durch einige Aufrufe der beiden Funktionen mit unterschiedlichen Mengen an Zahlen (Datei `ub58.php`). Diese Bibliothek können Sie später erweitern und auch für andere Programme nutzen.

B.8 Beispiele

In diesem Abschnitt finden sich einige umfangreichere Beispiele, in denen Sie Ihre Kenntnisse aus dem Programmierkurs anwenden können. Sie beinhalten keine neuen Programmierelemente, sondern dienen zur Darstellung des Zusammenspiels der verschiedenen Elemente.

Zur übersichtlichen und einheitlichen Darstellung von Zahlen (zum Beispiel in einigen Tabellen dieses Abschnitts) wird allerdings zunächst die Formatierung von Zahlen eingeführt.

B.8.1 Formatierung von Zahlen

Die formatierte Ausgabe von Zahlen wird mit Hilfe der Funktion `number_format()` durchgeführt. Ein Beispiel: number_format()

```
<html>
<body>
<?php
   echo "<b>Zahlen-Formatierung:</b><br>";
   $d = 12.3 * 3098.55397 * 445.2;
   echo "Variable d: $d<br>";
   echo "Mit Tausender-Teilung (englisch),<br> ohne
      Dezimalstellen: " . number_format($d) . "<br>";
   echo "Mit Tausender-Teilung, <br> auf drei
        Dezimalstellen gerundet (englisch): "
        . number_format($d,3) .
      "<br>";
   echo "Mit Tausender-Teilung,<br> auf drei
        Dezimalstellen gerundet (deutsch): "
        . number_format($d,3,",",".") . "<p>";
?>
</body>
</html>
```

Listing B.44 Datei ub59.php

Die Funktion `number_format()` kann mit einem, zwei oder vier Parametern aufgerufen werden:

- Falls sie mit einem Parameter aufgerufen wird, wird die Zahl mit Kommata als Tausendertrennzeichen ausgegeben, und zwar ohne Nachkommastellen.

- Falls sie mit zwei Parametern aufgerufen wird, wird die Zahl mit Kommata als Tausender-Trennzeichen ausgegeben, und zwar mit der Anzahl an Nachkommastellen, die im zweiten Parameter angegeben ist.
- Falls sie mit vier Parametern aufgerufen wird, wird die Zahl mit dem vierten Parameter als Tausender-Trennzeichen, der gewünschten Anzahl an Nachkommastellen und dem dritten Parameter als Dezimaltrennung ausgegeben.

Die Bildschirmausgabe hat folgendes Aussehen:

Abbildung B.63 Formatierung von Zahlen

B.8.2 Geldanlage

Der Benutzer besucht die Website einer Bank, die verschiedene Möglichkeiten zur Geldanlage bietet. Eine dieser Möglichkeiten ist die Anlage eines bestimmten Betrages über eine festgelegte Laufzeit. Je länger das Geld angelegt wird, desto höher ist der Zinssatz. Der Benutzer gibt den angelegten Betrag und die Laufzeit ein und bekommt als Antwort eine Tabelle, in der die Entwicklung seiner Geldanlage von Jahr zu Jahr dargestellt wird.

Der Zinssatz in Abhängigkeit von der Laufzeit:

```
<= 3 Jahre   3 %
<= 5 Jahre   4 %
<=10 Jahre   5 %
>10 Jahre    6 %
```

Das Eingabeformular:

```
<html>
<body>
<h2>Geldanlage</h2>
Geben Sie bitte die folgenden Werte ein<p>
<form action="ub60.php" method="post">
<input name="grundbetrag"> Grundbetrag (in Euro)<p>
<input name="laufzeit"> Laufzeit (in Jahren)<p>
<input type="submit"><p>
<input type="reset">
</form>
</body>
</html>
```

Listing B.45 Datei ub60.htm

Das Formular sieht wie folgt aus (mit Beispieldaten):

Abbildung B.64 Eingabeformular Geldanlage

Im PHP-Auswertungsprogramm wird zunächst mit einer mehrfachen Verzweigung aus der Laufzeit der Zinssatz bestimmt. Anschließend wird eine Schleife durchlaufen, pro Jahr der Geldanlage ein Durchlauf. Bei jedem Durchlauf wird der bis dahin entstandene Gesamtbetrag berechnet, formatiert und ausgegeben. Das Programm:

```
<html>
<body>
```

```
<h2>Geldanlage</h2>

<table border>
<tr>
<td align="right"><b>nach Jahr</b></td>
<td align="right"><b>Betrag</b></td>
</tr>

<?php
/* Zinssatz in Abhängigkeit der Laufzeit */
if ($laufzeit <= 3)
   $zinssatz = 3;
else if ($laufzeit <= 5)
   $zinssatz = 4;
else if ($laufzeit <= 10)
   $zinssatz = 5;
else
   $zinssatz = 6;

/* Anlageberechnung und Ausgabe */
$betrag = $grundbetrag;
for($i=1; $i<=$laufzeit; $i=$i+1)
{
   echo "<tr>";
   echo "<td align='right'>$i</td>";
   $betrag = $betrag + $betrag * $zinssatz / 100;
   $ausgabe = number_format($betrag,2,",",".");
   echo "<td align='right'>$ausgabe Euro</td>";
   echo "</tr>";
}
?>

</table>
</body>
</html>
```

Listing B.46 Datei ub60.php

Die Ausgabe des oben angegebenen Beispiels sieht wie folgt aus:

Abbildung B.65 Ausgabe Geldanlage

B.8.3 Steuertabelle

Es soll eine (stark vereinfachte) Berechnung und Ausgabe von Steuersätzen, Steuerbeträgen und Netto-Einkommen vorgenommen werden. Der Steuersatz wird aus dem Brutto-Einkommen nach folgender Tabelle berechnet:

Brutto-Einkommen	Steuersatz
<= 12000	12 %
<= 20000	15 %
<= 30000	20 %
> 30000	25 %

Der Benutzer hat die Möglichkeit zur Eingabe der folgenden Daten:

- Startwert (erster Wert, für den die genannten Beträge berechnet werden),
- Endwert (letzter Wert, für den die genannten Beträge berechnet werden),
- Intervall (Abstand der einzelnen Werte voneinander).

Das Eingabeformular:

```
<html>
<body>
<h2>Steuertabelle</h2>
Geben Sie bitte die folgenden Werte ein<p>
<form action="ub61.php" method="post">
<input name="start"> Startwert (in &euro;)<p>
<input name="ende"> Endwert (in &euro;)<p>
<input name="intervall"> Intervall (in &euro;)<p>
<input type="submit"><p>
<input type="reset">
</form>
</body>
</html>
```

Listing B.47 Datei ub61.htm

Das Euro-Zeichen wird mit der HTML-Entity `&euro;` erzeugt. Das Formular sieht wie folgt aus:

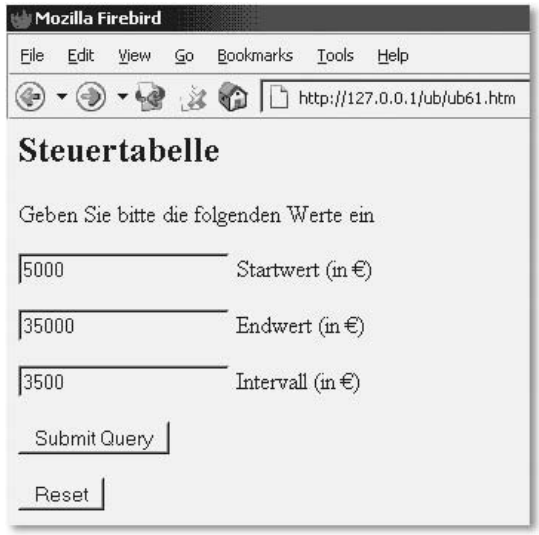


Abbildung B.66 Eingabeformular Steuertabelle

Im PHP-Auswertungsprogramm wird eine Schleife durchlaufen, pro Brutto-Einkommen ein Durchlauf. Innerhalb der Schleife wird zunächst mit einer mehrfachen Verzweigung aus dem Brutto-Einkommen der Steuersatz bestimmt. Abhängig vom Steuersatz werden Steuer-

betrag und Netto-Einkommen berechnet. Alle vier Informationen werden formatiert und ausgegeben. Die Ausgabe bietet eine Tabelle mit vier Spalten:

- Brutto-Einkommen in Euro,
- Steuersatz in Prozent,
- Steuerbetrag in Euro,
- Netto-Einkommen in Euro.

Das Programm:

```
<html>
<body>
<h2>Steuertabelle</h2>

<table border>
<tr>
<td align="center"><b>Gehalt</b></td>
<td align="center"><b>Steuersatz</b></td>
<td align="center"><b>Steuerbetrag</b></td>
<td align="center"><b>Netto</b></td>
</tr>

<?php
for($brutto = $start; $brutto <= $ende;
    $brutto = $brutto + $intervall)
{
   /* Berechnung des Steuersatzes */
   if($brutto <= 12000)
      $satz = 12;
   else if($brutto <= 20000)
      $satz = 15;
   else if($brutto <= 30000)
      $satz = 20;
   else
      $satz = 25;

   $steuerbetrag = $brutto * $satz / 100;
   $netto = $brutto - $steuerbetrag;
   echo "<tr>";
   echo "<td align='right'>" .
```

```
            number_format($brutto,2,",",".") .
            " &euro;</td>";
    echo "<td align='right'>" .
            number_format($satz,1,",",".") . " %</td>";
    echo "<td align='right'>" .
            number_format($steuerbetrag,2,",",".") .
            " &euro;</td>";
    echo "<td align='right'>" .
            number_format($netto,2,",",".") .
            " &euro;</td>";
    echo "</tr>";
}
?>

</table>
</body>
</html>
```

Listing B.48 Datei ub61.php

Die Ausgabe des oben angegebenen Beispiels:

Gehalt	Steuersatz	Steuerbetrag	Netto
5.000,00 €	12,0 %	600,00 €	4.400,00 €
8.500,00 €	12,0 %	1.020,00 €	7.480,00 €
12.000,00 €	12,0 %	1.440,00 €	10.560,00 €
15.500,00 €	15,0 %	2.325,00 €	13.175,00 €
19.000,00 €	15,0 %	2.850,00 €	16.150,00 €
22.500,00 €	20,0 %	4.500,00 €	18.000,00 €
26.000,00 €	20,0 %	5.200,00 €	20.800,00 €
29.500,00 €	20,0 %	5.900,00 €	23.600,00 €
33.000,00 €	25,0 %	8.250,00 €	24.750,00 €

Abbildung B.67 Ausgabe Steuertabelle

B.8.4 Bestimmung des Ostersonntags

Ostersonntag

In diesem Abschnitt soll eine Funktion `ostersonntag()` zur Bestimmung des Termins des Ostersonntags in einem vorgegebenen Jahr entwickelt werden. Auf Basis des Ostersonntags können alle beweglichen Feiertage eines Bundeslandes berechnet werden. Eine Liste der (beweglichen und festen) Feiertage wird häufig im Zusammenhang mit Terminplanungsprogrammen benötigt (siehe Beispiel in Abschnitt F.4, Datum und Zeit).

Die Funktion `ostersonntag()` soll in der Funktionsbibliothek `ub62datum.inc.php` bereitgestellt werden. Sie soll mit Hilfe eines Formulars (Datei `ub62.htm`) und eines PHP-Programms (Datei `ub62.php`) getestet werden. Im Formular werden vom Benutzer in zwei Eingabefeldern zwei Jahreszahlen angegeben. Das Programm liefert eine Tabelle, in der zu jedem Jahr im angegebenen Jahresbereich der jeweilige Termin des Ostersonntags ausgegeben wird.

Falls der Benutzer zum Beispiel die folgende Eingabe vornimmt:

Abbildung B.68 Eingabe Jahresbereich

wird die folgende Tabelle geliefert:

Abbildung B.69 Ausgabe der Ostersonntage im Bereich

Zur Bestimmung des Ostersonntags:

Ostern ist stets am ersten Sonntag nach dem ersten Vollmond des Frühlings. So hat es das erste Kirchenkonzil im Jahre `325` festgelegt, und dies gilt bis heute. Im Jahre `1800` entwickelte der große deutsche Mathematiker Carl Friedrich Gauß (`1777` bis `1855`) eine Formel zur Berechnung des Ostersonntags. Sie ist so genau, dass erst im Jahre `8202` ein Fehler auftritt.

Die Formel des Mathematikers: Ostern fällt im Jahre `J` auf den (`e+D+1`)ten Tag nach dem `21. März`, wobei gilt:

- `e = (2 * (J mod 4) + 4 * (J mod 7) + 6 * D + (6 + J/100 - J/400 - 2) mod 7) mod 7`.
- Falls `d = 29`, dann ist `D = 28`.
- Falls `d = 28` und `J mod 17 >= 11`, dann ist `D = 27`.
- Falls `d` weder `28` noch `29`: dann ist `D = d`.
- `d = ((15 + J/100 - J/400 - (8 * J/100 + 13) / 25) mod 30 + 19 * (J mod 19)) mod 30`.

Zur Umsetzung in ein Programm muss man wissen:

- `mod` entspricht dem Modulo-Operator aus PHP. Dies ist also der Rest einer Division.
- Alle vorkommenden Divisionen (zum Beispiel `J/100`) sind Ganzzahldivisionen, die Stellen hinter dem Komma werden abgeschnitten. Zum Abschneiden kann man die mathematische Funktion `floor()` benutzen. Der Ausdruck `1952/100` ergibt den Wert `19.52` (mit Nachkommastellen). Der Ausdruck `floor(1952/100)` ergibt den Wert `19` (ohne Nachkommastellen, also Ganzzahldivision).

Die Funktion `ostersonntag()` in der Bibliothek ergibt sich wie folgt:

```
<?php
function ostersonntag($j, $t, $m)
{
   // Berechnung von klein d
   $d = ((15 + floor($j/100) - floor($j/400)
      - floor((8 * floor($j/100) + 13) / 25)) % 30
      + 19 * ($j % 19)) % 30;

   // Berechnung von groß D
   if ($d==29)
      $D = 28;
   else if ($d == 28 && $j%17 >= 11)
      $D = 27;
   else
      $D = $d;

   // Berechnung von klein e
   $e = (2 * ($j%4) + 4 * ($j%7) + 6 * $D
      + (6+floor($j/100)-floor($j/400) - 2) % 7) % 7;

   // Berechnung von Tag und Monat
   // Rückgabe der Werte per Referenz
   $m = "03";
   $t = 21 + $e + $D + 1;
   if ($t > 31)
   {
      $m = "04";
      $t = $t - 31;
   }
```

```
   if($t < 10)
      $t = "0" . $t;
}
?>
```

Listing B.49 Datei ub62datum.inc.php

Das Jahr wird über den Parameter `$j` an die Funktion geliefert. `$t` und `$m` sind zwei Referenzen für die beiden Variablen für Tag und Monat. Diese beiden Werte stehen nach Aufruf der Funktion an der Aufrufstelle zur Verfügung. Innerhalb der Funktion wird das Ergebnis in einzelnen Schritten bestimmt:

- Der Wert von `$d` wird gemäß der oben angegebenen Formel errechnet.
- Der Wert von `$D` ergibt sich mit Hilfe einer mehrfachen Verzweigung aus `$d`.
- Der Wert von `$e` wird gemäß der oben angegebenen Formel errechnet.
- Falls der errechnete Tag nicht mehr im Monat März liegt, müssen Tag und Monat auf den entsprechenden Tag im Monat April umgerechnet werden. Beispiel: aus dem `36.03.` wird der `05.04.`
- Die Zahlen werden in Text umgewandelt, mit führenden Nullen bei den einstelligen Zahlen.

Das Eingabeformular für den Benutzer:

```
<html>
<body>
<h2>Ostersonntag</h2>
Bitte geben Sie zwei Jahreszahlen ein<br>
und senden Sie das Formular ab:<p>
<form action="ub62.php" method="post">
    <input name="anfang"> Erste Jahreszahl<p>
    <input name="ende"> Zweite Jahreszahl<p>
    <input type="submit">
    <input type="reset">
</form>
</body>
</html>
```

Listing B.50 Datei ub62.htm

Die beiden Jahreszahlen werden in den Feldern `anfang` und `ende` eingegeben.

Das PHP-Programm zur Erzeugung der Tabelle:

```
<html>
<body>
<h2>Ostersonntag</h2>
<?php
   // Einbinden der Funktionsbibliothek
   include "ub62datum.inc.php";

   // Größere Jahreszahl zuerst? Tauschen!
   if ($anfang > $ende)
   {
      $temp = $anfang;
      $anfang = $ende;
      $ende = $temp;
   }
   echo "<table border>";
   echo "<tr> <td><b>Jahr</b></td>
        <td><b>Datum</b></td> </tr>";

   // Schleife über alle Jahreszahlen
   for ($j=$anfang; $j<=$ende; $j++)
   {
      ostersonntag($j, &$tag, &$monat);
      echo "<tr> <td>$j</td>
           <td>$tag.$monat.$j</td> </tr>";
   }
   echo "</table>";
?>
</body>
</html>
```

Listing B.51 Datei ub62.php

Die Funktionsbibliothek wird eingebunden, damit steht die Funktion `ostersonntag()` zur Verfügung. Falls der Benutzer die beiden Jahreszahlen in der falschen Reihenfolge eingegeben hat, werden sie getauscht. In einer Schleife wird die Funktion `ostersonntag()` für jeden Wert von `anfang` bis `ende` aufgerufen. In den beiden Variablen

`$tag` und `$monat` sind per Referenz nach jedem Aufruf die Werte für Tag und Monat des betreffenden Jahres gespeichert. Diese beiden Werte werden ausgegeben.

Index

B

C

D

E

F

G

H

I

J

K

P

R

S

T

U

V

W

Z