edv-buchversand.de

Einstieg in PHP 8 und MySQL

Ideal für Programmieranfänger

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

Kapitel 1
PHP-Programmierkurs

In diesem Kapitel lernen Sie, mithilfe von Variablen, Feldern, Operatoren, Kontrollstrukturen und Funktionen erfolgreich Programme in PHP zu schreiben. Die Auswertung von Formularen und einige umfangreichere Beispiele runden das Kapitel ab.

Hinweis

Dieses Buch soll Ihnen nicht nur Kenntnisse der Sprache PHP vermitteln, sondern Ihnen auch einen übersichtlichen und strukturierten Programmierstil beibringen. Er vereinfacht sowohl die Arbeit eines einzelnen Entwicklers als auch die Zusammenarbeit eines Entwicklerteams und die spätere Wartung der Programme.

Für viele denkbare Anwendungsfälle biete ich jeweils nur eine Lösung an und erläutere den typischen Einsatzzweck, denn ich möchte Sie nicht durch eine allzu große Anzahl von Möglichkeiten verwirren.

[«]

1.1 Einbettung von PHP

Die Markierung `<?php` leitet eine oder mehrere PHP-Anweisungen ein. Diese werden bis zur Markierung `?>` bearbeitet, die das Ende des PHP-Bereichs darstellt. PHP-Bereiche können im gesamten Dokument untergebracht werden. Der Code wird von oben nach unten abgearbeitet. Dabei kann mehrmals zwischen HTML und PHP gewechselt werden.

Zur Auffrischung Ihrer Kenntnisse in HTML und CSS verweise ich auf Abschnitt A.5. In ihm werden einige Grundlagen erläutert, die bei der Programmierung mit PHP benötigt werden.

Das folgende Beispiel zeigt die Einbettung von PHP-Code in HTML:

```
<!DOCTYPE html>
<html lang="de">
<head>
   <meta charset="utf-8">
   <title>PHP einbetten</title>
```

```
</head>
<body>
Die erste Zeile in HTML<br>
<?php
   echo "Die zweite Zeile in PHP<br>";
   echo "Die dritte " . "Zeile in PHP" . "<br>";
?>
</body>
</html>
```

Listing 1.1 Die Datei »einbetten.php«

Die PHP-Anweisung `echo` gibt den angegebenen Text auf dem Bildschirm aus. Der Text muss in einfachen oder in doppelten Hochkommata geschrieben werden. Enthält der Text HTML-Markierungen (hier `<br>` für einen Zeilenumbruch), werden diese im Browser ausgeführt. Der Text kann aus mehreren Teilen bestehen. Diese müssen mithilfe des Operators `.` (Punkt) miteinander verbunden werden.

Die Ausgabe des Programms sehen Sie in Abbildung 1.1.

Abbildung 1.1 Einbetten von PHP in HTML

Um das Beispiel nachzuvollziehen, gehen Sie wie folgt vor:

- Starten Sie den Apache-Webserver, wie es bei der Installation von *XAMPP* im Anhang beschrieben wird.
- Geben Sie den angegebenen Code in einem Editor ein, und speichern Sie ihn in der Datei *einbetten.php* im Hauptverzeichnis des Webservers. Das jeweils passende Verzeichnis auf der Festplatte Ihres Rechners wird ebenfalls im Anhang genannt.

Die vollständige Adresse des Webservers ist *http://localhost*. Die Funktionsweise der PHP-Dateien wird über den Webserver kontrolliert. Daher lautet die vollständige Adresse für das erste Beispielprogramm *http://localhost/einbetten.php*. In der Adresszeile eines modernen Browsers genügt meist die Eingabe ohne den Protokollnamen, also *localhost/einbetten.php*.

Sollten Sie in Ihrem Browser nicht die gleiche Ausgabe wie in Abbildung 1.1 sehen, kann dies mehrere Ursachen haben. Prüfen und korrigieren Sie gegebenenfalls Ihren Programmcode und die eingegebene Adresse.

Beim Programmieren arbeitet man häufig mit Einrückungen. Damit wird die Struktur eines Dokuments sowohl im HTML-Code als auch im PHP-Code besser erkennbar.

1.1.1 Kodierung in UTF-8

In den Dokumenten dieses Buches wird mithilfe der Meta-Angabe `charset` die Kodierung UTF-8 eingestellt. Es ist wichtig, dass die Kodierung, die im `head`-Container steht (siehe Listing 1.1), mit der Kodierung der Datei übereinstimmt. Steht die Kodierung einer Datei im Editor Notepad++ noch nicht auf UTF-8, können Sie sie wie folgt umstellen: Menüpunkt KODIERUNG • KONVERTIERE ZU UTF-8. Anschließend ist innerhalb des Menüs KODIERUNG die Kodierung UTF-8 markiert.

Sie können die Kodierung im Editor Notepad++ wie folgt auch automatisch für alle Dateien wählen, die Sie neu erstellen: Menüpunkt EINSTELLUNGEN • OPTIONEN • NEUE DATEIEN • KODIERUNG • UTF-8. Betätigen Sie als Letztes die Schaltfläche SCHLIESSEN.

UTF-8 ist die Abkürzung für das *8-Bit UCS Transformation Format*. *UCS* steht für *Universal Character Set*. UTF-8 ist die Kodierung mit der weitesten Verbreitung für Unicode-Zeichen. Sie enthält auch viele Sonderzeichen, zum Beispiel die deutschen Umlaute und das *ß*.

1.1.2 Kommentare

Durch Kommentare wird ein Programm lesbarer. Sie werden nicht ausgeführt, sondern dienen nur zur Information der Personen, die das Programm entwickeln, insbesondere bei umfangreichen Programmen. Sollte es sich um eine Gruppe von Entwicklern und Entwicklerinnen handeln oder sollte das Programm später von anderen Personen weiterbearbeitet werden, ist es besonders notwendig, Kommentare zu schreiben.

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 Kolleginnen und Kollegen heraus.

Man unterscheidet zwischen einzeiligen und mehrzeiligen Kommentaren:

- Ein einzeiliger Kommentar beginnt mit den Zeichen // und endet am Schluss 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 Programmblocks verwendet.

Ein Beispiel hierzu:

```
<!DOCTYPE html>...<body>
<?php
   echo "Das ist der Anfang";    // Kommentar
                                 // bis zum Zeilenende
   /* Ein Kommentar in
      mehreren Zeilen */
   echo " und hier das Ende des Programms.";
?>
</body></html>
```

Listing 1.2 Die Datei »kommentar.php«

Die Ausgabe des Programms im Browser sehen Sie in Abbildung 1.2. In diesem und in vielen nachfolgenden Codebeispielen bleibt der Beginn des HTML-Dokuments bis auf den Titel gleich. Daher wird er für die Darstellung im Buch häufig mithilfe von ... verkürzt.

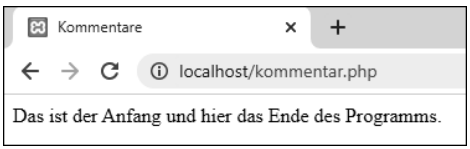


Abbildung 1.2 Ausgabe ohne Kommentare

[ ] **Übung »u_ausgabe«**

Schreiben Sie ein PHP-Programm innerhalb einer Webseite (Datei *u_ausgabe.php*) mit Kommentarzeilen. Speichern Sie die Datei im Hauptverzeichnis Ihres Webservers, und testen Sie das Programm, indem Sie einen Browser aufrufen und die Adresse *localhost/u_ausgabe.php* eingeben. Die Ausgabe des Programms im Browser sollte so aussehen wie in Abbildung 1.3.

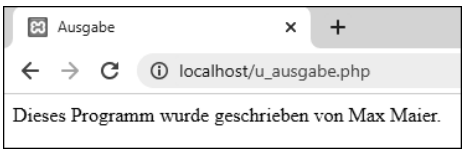


Abbildung 1.3 Ergebnis der Übung »u_ausgabe«

1.2 Variablen, Datentypen und Operatoren

Innerhalb eines Programms können Informationen zur späteren Verwendung in Variablen gespeichert werden.

1.2.1 Datentypen

Variablen unterscheiden sich in ihren Datentypen. PHP unterstützt unter anderem Datentypen für:

- ganze Zahlen
- *Fließkommazahlen*, also Zahlen mit Nachkommastellen
- Zeichenketten (auch *Strings* genannt)
- Felder (ein- und mehrdimensionale Felder von Variablen)
- Objekte

Der Datentyp für eine PHP-Variable wird nicht beim Programmieren festgelegt, sondern richtet sich nach dem Zusammenhang, in dem die Variable verwendet wird. Sie kann bei ihrem ersten Erscheinen sofort benutzt werden und muss dem Programm nicht vorher bekannt gemacht werden. Sie kann ihren Datentyp innerhalb eines Programms wechseln.

Seit PHP 7.0 gibt es die Möglichkeit, die Datentypen der benutzten Variablen beim Aufruf von Funktionen genauer zu prüfen. Dabei wird mit sogenannten Typhinweisen gearbeitet. Typhinweise verbessern die Lesbarkeit und erleichtern den Ablauf und die Pflege von PHP-Programmen. Mehr dazu finden Sie in Abschnitt 1.10.11.

Zunächst geht es um die *einfachen Datentypen* (Zahlen und Zeichenketten). Später kommen Felder und Objekte hinzu.

1.2.2 Namen für Variablen

Für den Namen einer Variablen gelten folgende Regeln:

- Er muss mit einem Dollarzeichen beginnen.
- Er darf keine Leerzeichen enthalten.
- Er darf nur aus Buchstaben und Ziffern bestehen, wobei das erste Zeichen ein Buchstabe sein muss. Es sind Groß- und Kleinbuchstaben erlaubt, zwischen denen jedoch unterschieden wird (`$HokusPokus` ist nicht das Gleiche wie `$hokuspokus`).

- Er darf keine sprachspezifischen Zeichen wie zum Beispiel die deutschen Umlaute oder ß (»scharfes S«) enthalten.
- Er darf als einziges Sonderzeichen den _ (Unterstrich) enthalten.
- Er darf nicht mit einem reservierten Wort identisch sein, zum Beispiel mit einem Befehl aus der Sprache PHP.

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

Diese Regeln gelten in ähnlicher Form für die Namen von Konstanten (siehe Abschnitt 1.2.8), Funktionen (siehe Abschnitt 1.7) sowie Klassen und Methoden (siehe Kapitel 4). Eine wichtige Ausnahme: Nur die Namen von Variablen beginnen mit einem Dollarzeichen.

1.2.3 Variablen für Zahlen

Betrachten Sie einmal das folgende Programm, in dem verschiedene Zahlen gespeichert und ausgegeben werden:

```
<!DOCTYPE html>...<body>
<?php
   $ganzeZahl = -14;
   $kommaZahl = 1.35;
   $grosseZahl = 5.5e6;
   $kleineZahl = 3.8e-3;
   $separatorGross = 1_580_000_000;
   $separatorKlein = 0.000_000_001_580;

   echo $ganzeZahl . "<br>";
   echo $kommaZahl . "<br>";
   echo $grosseZahl . "<br>";
   echo $kleineZahl . "<br>";
   echo $separatorGross . "<br>";
   echo $separatorKlein;
?>
</body></html>
```

Listing 1.3 Die Datei »zahl_variable.php«

Es wird eine Variable mit dem Namen `$ganzeZahl` eingeführt. Dieser Variablen wird der Wert `-14` zugewiesen, wodurch `$ganzeZahl` zu einer Variablen für eine ganze Zahl wird. Bei negativen Werten wird ein Minuszeichen vorangestellt.

Die Variable `$kommaZahl` wird eingeführt. Ihr wird der Wert `1.35` zugewiesen, also wird `$kommaZahl` zu einer Variablen für eine Fließkommazahl. Dabei muss die englische Schreibweise mit einem Punkt als Dezimaltrennzeichen verwendet werden.

Eine Zahl kann auch als *Exponentialzahl* eingegeben werden. Diese Schreibweise eignet sich besonders für sehr große oder sehr kleine Zahlen:

- Die Variable `$grosseZahl` erhält den Wert `5.5e6`. Er wird gelesen als `5.5 × 10 hoch 6`, also `5.5 × 1000000`. Es ergibt sich der Wert `5500000`.
- Die Variable `$kleineZahl` erhält den Wert `3.8e-3`. Er wird gelesen als `3.8 × 10 hoch -3`, also `3.8 × 0.001`. Es ergibt sich der Wert `0.0038`.

Seit PHP 7.4 kann ein Unterstrich als *Separator* dienen. Ein Separator hilft bei der Eingabe von Zahlen mit vielen Ziffern, kann an beliebiger Stelle eingesetzt werden und hat keinen Einfluss auf den Zahlenwert. Lange Ziffernfolgen werden meist in Dreiergruppen unterteilt.

Mit `echo` lassen sich nicht nur Texte, sondern auch die gespeicherten Werte von Variablen ausgeben. Auch hier gilt: Mehrere Teile der Ausgabe werden mithilfe eines Punkts miteinander verbunden. Abbildung 1.4 zeigt die Ausgabe des Programms im Browser.

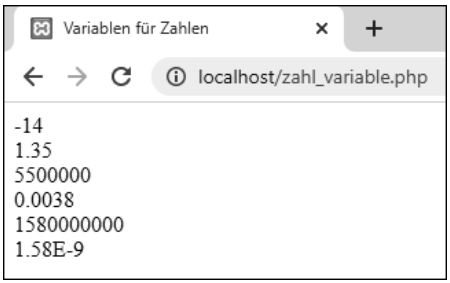


Abbildung 1.4 Zahlen

1.2.4 Rechenoperatoren für Zahlen

Bei Zahlen können Sie die Rechenoperatoren (arithmetischen Operatoren) aus Tabelle 1.1 verwenden.

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo-Operation: der Rest bei einer ganzzahligen Division
**	Potenzieren mithilfe des Exponentialoperators. Ein Beispiel: `2 ** 3`, gesprochen: »2 hoch 3«. Mehr dazu folgt in Abschnitt 10.2.

Tabelle 1.1 Rechenoperatoren in PHP

Ein Beispiel mit einigen Berechnungen:

```
<!DOCTYPE html>...<body>
<?php
   $zahlEins = 3 * 2 + 2.5 * 4 - 3;
   $zahlZwei = 3 * (2 + 2.5) * 4 - 3;
   $zahlDrei = 23 / 4;
   $zahlVier = 23 % 4;

   echo $zahlEins . "<br>";
   echo $zahlZwei . "<br>";
   echo $zahlDrei . "<br>";
   echo $zahlVier;
?>
</body></html>
```

Listing 1.4 Die Datei »zahl_operator.php«

Beachten Sie die Rangfolge der Operatoren: Die Operatoren für die Multiplikation und die Division haben wie in der Mathematik Vorrang vor den Operatoren für die Addition und die Subtraktion. Sie werden also zuerst ausgeführt. Bei Operatoren mit gleicher Rangfolge werden die Ausdrücke von links nach rechts bearbeitet.

Ausdrücke in Klammern werden vorrangig berechnet. Daher ergeben sich für die ersten beiden Rechenausdrücke unterschiedliche Ergebnisse.

Die mathematische Division von 23 durch 4 ergibt 5.75. Die ganzzahlige Division von 23 durch 4 ergibt 5 Rest 3. Dieser Rest wird mithilfe des Modulo-Operators ermittelt.

Hinweis [«]

Eine (mathematisch nicht erlaubte) Division einer positiven oder negativen Zahl durch 0 führt zur Ausgabe einer Warnung. Seit PHP 7.0 führt sie nicht mehr zu einem Abbruch des Programms. Als Ergebnis wird `INF` beziehungsweise `-INF` angezeigt. INF steht als Abkürzung für *infinity* (deutsch: *unendlich*).

Die Ausgabe des Programms sehen Sie in Abbildung 1.5.

Abbildung 1.5 Berechnungen mit Zahlen

Übung »u_zahl« [✎]

Berechnen Sie in einem PHP-Programm (Datei *u_zahl.php*) den Bruttopreis eines Einkaufs. Es werden insgesamt drei Artikel eingekauft. Die Nettopreise der einzelnen Artikel betragen 22,50 €, 12,30 € und 5,20 €. Der Bruttopreis berechnet sich bekanntlich aus dem Nettopreis zuzüglich 19 % Umsatzsteuer. In die Berechnung muss also der Faktor `1.19` eingehen.

Ihr Programm sollte im Browser über die Adresse *localhost/u_zahl.php* aufrufbar sein. Die Ausgabe sollte so wie in Abbildung 1.6 aussehen.

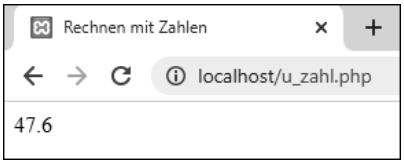


Abbildung 1.6 Ergebnis der Übung »u_zahl«

1.2.5 Kombinierte Zuweisungsoperatoren

Sie kennen bereits den Zuweisungsoperator `=`. Es gibt zudem die kombinierten Zuweisungsoperatoren `+=`, `-=`, `*=`, `/=`, `%=` und `**=`, mit deren Hilfe eine Operation mit einer Zuweisung kombiniert werden kann.

Der *Inkrement-Operator* ++ dient zur Erhöhung einer Zahl um 1. Das Gegenstück ist der *Dekrement-Operator* -- zur Verminderung einer Zahl um 1. In diesem Buch werden die beiden Operatoren normalerweise nur bei alleinstehenden Variablen eingesetzt, zum Beispiel bei `for`-Schleifen, siehe auch Abschnitt 1.6.

Nachfolgend sehen Sie unter anderem die Auswirkungen, falls die beiden Operatoren in Zuweisungen verwendet werden:

```
<!DOCTYPE html>...<body>
<?php
   $zahl = 5;     echo $zahl . " ";
   $zahl += 7;    echo $zahl . " ";
   $zahl -= 7;    echo $zahl . " ";
   $zahl *= 3;    echo $zahl . " ";
   $zahl /= 3;    echo $zahl . " ";
   $zahl %= 3;    echo $zahl . "<br>";

   $a = 5;        echo $a . " ";
   $a++;          echo $a . " ";
   ++$a;          echo $a . "<br>";

   $a = 5;
   $b = $a++;
   echo $a . " " . $b . "<br>";

   $a = 5;
   $b = ++$a;
   echo $a . " " . $b;
?>
</body></html>
```

Listing 1.5 Die Datei »zahl_zuweisung.php«

Die Variable `$zahl` hat zunächst den Wert 5. Er wird um 7 erhöht und anschließend um 7 vermindert. Danach wird er mit 3 multipliziert und anschließend durch 3 geteilt. Als Letztes wird der Wert der Zahl zugewiesen, der sich bei der Modulo-Operation ergibt.

Der Inkrement-Operator ++ kann vor (*Präfix-Notation*) oder nach (*Postfix-Notation*) einer Variablen stehen. Steht die Variable allein, ist das Ergebnis identisch: Die Variable wird um 1 erhöht.

Steht die Variable nicht allein, sondern innerhalb einer Zuweisung, ist die Reihenfolge der Bearbeitung zu beachten. Bei `$b = $a++` wird zunächst der alte Wert von `$a` an `$b` über-

geben. Anschließend wird $a erhöht. Bei $b = ++$a wird zunächst $a erhöht. Anschließend wird der neue Wert von $a an $b übergeben. Beim Dekrement-Operator -- zur Verminderung einer Variablen um 1 verhält es sich entsprechend.

Die Ausgabe des Programms sehen Sie in Abbildung 1.7.

Abbildung 1.7 Kombinierte Zuweisungen

1.2.6 Formatierung von Zahlen

PHP bietet eine ganze Reihe von Funktionen, die Sie für Ihre Zwecke nutzen können. Die beiden Funktionen `number_format()` und `sprintf()` liefern Zahlen, die man ihnen übergibt, formatiert zurück, sodass sie zum Beispiel auf dem Bildschirm ausgegeben werden können. Im folgenden Programm sehen Sie einige Beispiele:

```
<!DOCTYPE html>...<body>
<?php
   $zahl = 1234567.987654;
   echo $zahl . "<br>";
   echo number_format($zahl) . "<br>";
   echo number_format($zahl, 3) . "<br>";
   echo number_format($zahl, 3, ",", ".") . "<br><br>";

   echo sprintf("%.3f", $zahl) . "<br>";
   echo sprintf("%.3e", $zahl) . "<br>";

   $tag = 7;
   echo sprintf("%d", $tag) . "<br>";
   echo sprintf("%'.02d", $tag);
?>
</body></html>
```

Listing 1.6 Die Datei »zahl_formatierung.php«

In der Variablen $zahl ist ein Fließkommawert mit sechs Nachkommastellen gespeichert. In der ersten Ausgabe sehen Sie die Zahl mit allen Nachkommastellen.

Die Funktion `number_format()` kann mit einer unterschiedlichen Anzahl von Parametern aufgerufen werden. Parameter stehen in den runden Klammern nach dem Funktionsnamen und werden jeweils durch ein Komma voneinander getrennt:

- Wird die Funktion nur mit einem Parameter aufgerufen, wird der übergebene Wert ohne Nachkommastellen dargestellt, mit einem Komma als Tausender-Trennzeichen.
- Wird die Funktion mit zwei Parametern aufgerufen, wird im zweiten Parameter die Anzahl der Nachkommastellen angegeben, auf die gerundet wird.
- Wird die Funktion mit vier Parametern aufgerufen, dient der dritte Parameter zur Angabe des Dezimalzeichens (hier ist es ein Komma) und der vierte Parameter zur Angabe des Tausender-Trennzeichens (hier ist es der Punkt).

Die Funktion `sprintf()` wird mit einer Formatierungszeichenkette als erstem Parameter aufgerufen. Diese enthält die Angaben zur Formatierung der weiteren Parameter. Im vorliegenden Beispiel gibt es jeweils nur einen weiteren Parameter. Mögliche Angaben sind:

- `%f`: zur Ausgabe als Fließkommazahl
- `%.3f`: zur Rundung der Fließkommazahl auf die angegebene Anzahl von Nachkommastellen
- `%e`: zur Ausgabe als Exponentialzahl
- `%.3e`: zur Rundung der Exponentialzahl auf die angegebene Anzahl von Nachkommastellen
- `%d`: zur Ausgabe einer ganzen Zahl
- `%'.02d`: zur Ausgabe einer ganzen Zahl mit mindestens zwei Ziffern, gegebenenfalls mit der Ziffer 0 vorne aufgefüllt, wie es zum Beispiel für eine zweistellige Datumsangabe benötigt wird.

Die Ausgabe sehen Sie in Abbildung 1.8.

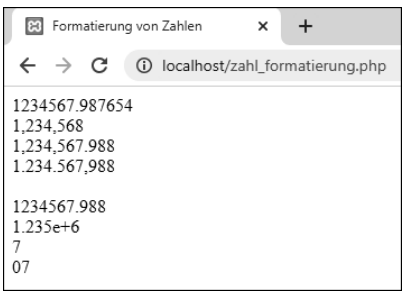


Abbildung 1.8 Formatierte Zahlen

1.2.7 Variablen und Operatoren für Zeichenketten

Einige Regeln und Operatoren für Zeichenketten werden anhand des nachfolgenden Beispiels erläutert:

```
<!DOCTYPE html>...<body>
<?php
   $a = 3;
   $b = 2.6;
   $c = $a + $b;

   echo "Summe: " . $c . "<br>";
   echo "Summe: " . ($a + $b) . "<br>";

   $zeile = "Addition: $a + $b = $c<br>";
   echo $zeile;

   $zeile = "Die Summe von";
   $zeile .= " $a + $b";
   $zeile .= " ist $c";
   $zeile .= "<br>";
   echo $zeile;
?>
</body></html>
```

Listing 1.7 Die Datei »zeichenkette.php«

Eine (verkettete) Zeichenkette kann in einer Variablen gespeichert werden oder unmittelbar mithilfe von `echo` ausgegeben werden. Die Variable `$zeile` wird durch die Zuweisung zu einer Variablen für eine Zeichenkette. Zeichenketten werden auch *Strings* genannt. Enthält eine Zeichenkette HTML-Code, gelangt dieser zur Ausführung.

Es können sowohl Zahlen als auch Zeichenketten mithilfe des Operators `.` (Punkt) miteinander verkettet werden. Findet dabei eine Berechnung statt, sollte diese in Klammern gesetzt werden, damit sie vor der eigentlichen Verkettung ausgeführt wird. Zur Verlängerung von Zeichenketten dient der Operator `.=`.

Der Name einer Variablen kann zur Vereinfachung direkt in eine Zeichenkette eingebettet werden. Gespeichert beziehungsweise ausgegeben wird der Wert der Variablen. Operatoren, die direkt in einer Zeichenkette eingebettet sind, gelangen nicht zur Ausführung, sondern werden als Text ausgegeben.

Die Ausgabe des Programms sehen Sie in Abbildung 1.9.

[»] **Hinweis**

Zeichenketten können auch innerhalb einfacher Hochkommata notiert werden. Ist eine Variable enthalten, wird allerdings ihr Name und nicht ihr Wert ausgegeben.

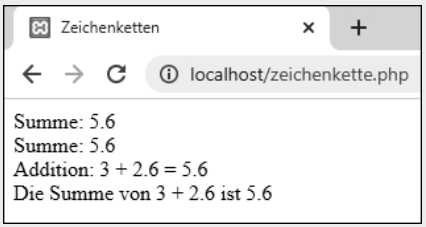


Abbildung 1.9 Arbeiten mit Zeichenketten

[✎] **Übung »u_zeichenkette«**

Schreiben Sie das Programm aus der vorherigen Übung *u_zahl* um (Datei *u_zeichenkette.php*). Die Zwischenergebnisse sollen einzeln berechnet und ausgegeben werden. Ihr Programm sollte im Browser über die Adresse *localhost/u_zeichenkette.php* aufrufbar sein. Die Ausgabe sollte so wie in Abbildung 1.10 aussehen.

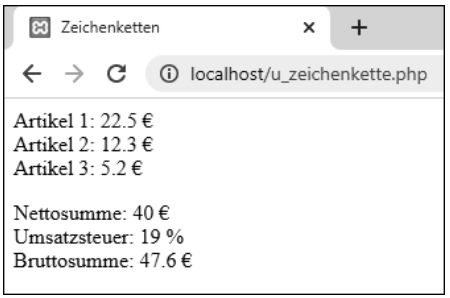


Abbildung 1.10 Das Ergebnis der Übung »u_zeichenkette«

1.2.8 Konstanten

Konstanten dienen zur Speicherung von unveränderlichen Werten. Beim Entwickeln können Sie sich den Namen einer Konstanten meist leichter merken als den zugehörigen Wert. Nachfolgend ein kleines Beispiel:

```
<!DOCTYPE html>...<body>
<?php
```

```
    const pi = 3.1415926;
    const gruss = "Guten Morgen";
    echo pi . "<br>";
    echo gruss . "<br>";
    // gruss = "Hallo";
?>
</body></html>
```

Listing 1.8 Die Datei »konstanten.php«

Mithilfe des Schlüsselworts `const` werden die Zahlenkonstante `pi` und die Zeichenkettenkonstante `gruss` definiert. Beachten Sie, dass im Unterschied zu Variablen kein `$`-Zeichen vor dem Namen notiert wird. Konstanten können nicht direkt innerhalb von Zeichenketten ausgegeben werden, da sie mangels `$`-Zeichen nicht vom restlichen Text unterschieden werden können. Beim Versuch, eine Konstante zu ändern, erscheint eine Fehlermeldung.

In Abbildung 1.11 sehen Sie die Ausgabe des Programms.

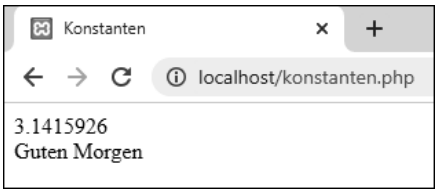


Abbildung 1.11 Konstanten

Hinweis [«]

Enumerationen sind, vereinfacht ausgedrückt, Aufzählungen von Konstanten. Durch die Nutzung der Elemente einer Enumeration wird der Programmcode besser lesbar und Vergleiche werden vereinfacht. Mehr zu Enumerationen finden Sie in Abschnitt 4.13.

1.2.9 Referenzen

Sie können auf eine vorhandene Variable eine sogenannte *Referenz* einrichten. Damit haben Sie die Möglichkeit, auf diese Variable über einen weiteren Namen zuzugreifen. Referenzen werden vor allem im Zusammenhang mit Funktionen und Methoden interessant (siehe Abschnitt 1.7.6). Eine erste Einführung bietet das folgende Programm:

```
<!DOCTYPE html>...<body>
<?php
   $orig = 12.3;
   echo "$orig<br>";

   $refe = &$orig;
   $refe = 5.8;
   echo "$orig<br>";
?>
</body></html>
```

Listing 1.9 Die Datei »referenz.php«

Die Variable `$orig` wird erzeugt. Es wird ihr ein Wert zugewiesen und ausgegeben. Danach wird mithilfe des Operators `&` die Referenz `$refe` auf die Variable `$orig` eingerichtet. Eine anschließende Änderung des Werts über die Referenz entspricht einer Änderung des Werts der Variablen. In Abbildung 1.12 sehen Sie die Ausgabe des Programms.

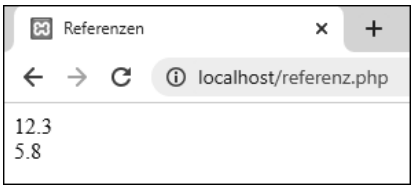


Abbildung 1.12 Referenzen

1.3 Einfache Formularauswertungen

In den bisher gezeigten Beispielen haben die Benutzerinnen eines Programms noch keine Möglichkeit, eigene Eingaben vorzunehmen. Sie können das Programm lediglich aufrufen und das Ergebnis betrachten.

Eine besondere Stärke und ein typischer Einsatzzweck von PHP ist jedoch die Auswertung von Benutzereingaben aus Formularen. Erst durch eine solche Auswertung wird die dynamische Informationsübermittlung zwischen den Benutzern und dem Webserver ermöglicht: Einem Betrachter wird zunächst ein Formular vorgelegt, in dem er eigene Einträge vornehmen kann beziehungsweise bei dem er aus bereits vorhandenen Einträgen auswählen kann. Er füllt das Formular aus, sendet es ab und erhält nach der Auswertung eine Antwort vom Webserver.

1.3.1 Eingabeformular

In diesem Abschnitt soll eine Informationsübermittlung mithilfe von einzeiligen Texteingabefeldern ermöglicht werden. Formulare können noch aus einer Reihe weiterer Elemente bestehen. Diese werden ausführlich in Kapitel 2 besprochen.

Der HTML-Programmcode des Formulars sieht so aus:

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

Listing 1.10 Die Datei »eingabe.htm«

Die Ausgabe des Formulars im Browser, mit eingegebenen Beispieldaten, sehen Sie in Abbildung 1.13.

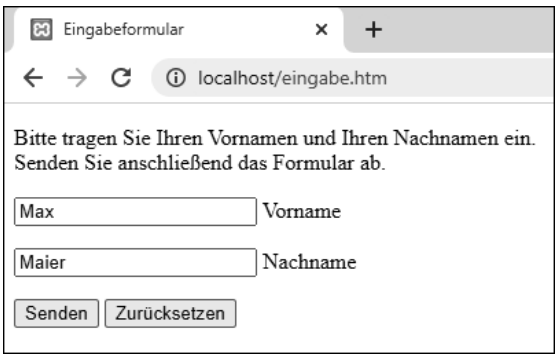


Abbildung 1.13 Eingabeformular mit Beispieldaten

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

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

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

- zwei einzeilige Texteingabefelder mit den Namen `vorname` und `nachname` für die Eintragung des Vor- und Nachnamens,
- eine Schaltfläche zum Absenden (englisch: *to submit*); beim Betätigen werden die eingetragenen Daten an den Server gesendet und es wird das genannte PHP-Auswertungsprogramm angefordert,
- eine Schaltfläche zum Zurücksetzen (englisch: *to reset*) des Formulars; beim Betätigen wird das Formular wieder in den Anfangszustand versetzt, wie es zum Beispiel bei einer Fehleingabe notwendig sein kann.

Falls Sie keinen eigenen Text vorgeben (siehe Abschnitt 2.1.1), wird auf den Schaltflächen ein Standardtext angezeigt, abhängig vom Browser und der Sprachversion. Die Auswertung der Eingabedaten stelle ich im folgenden Abschnitt vor.

[ ] **Übung »u_eingabe«, Teil 1**

Erweitern Sie das Beispiel dahingehend, dass eine vollständige Adresse eingegeben werden kann (Datei *u_eingabe.htm*). Es sollen zusätzlich drei weitere Eingabefelder für die Angaben zu Straße mit Hausnummer, Postleitzahl und Ort innerhalb des Formulars vorhanden sein. Das Formular sollte so wie in Abbildung 1.14 aussehen (mit Beispieldaten).

Abbildung 1.14 Erweitertes Eingabeformular mit Beispieldaten

1.3.2 Auswertung mit $_POST

Das antwortende PHP-Programm für das Formular sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   $vorname = htmlentities($_POST["vorname"]);
   $nachname = htmlentities($_POST["nachname"]);
   echo "Guten Tag, $vorname $nachname";
?>
</body></html>
```

Listing 1.11 Die Datei »eingabe.php«

Hat der Benutzer das obige Beispiel eingegeben, antwortet der Server so, wie in Abbildung 1.15 dargestellt.

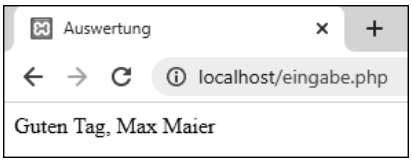


Abbildung 1.15 Auswertung des Eingabeformulars

Es gibt in PHP einige vordefinierte Variablen, unter anderem das assoziative Feld `$_POST`. Wird die Übermittlungsmethode `post` verwendet, werden aus den Namen der Eingabefelder automatisch Elemente des Felds `$_POST`.

Die Elemente können angesprochen werden, indem Sie ihren Namen in Hochkommata und eckigen Klammern hinter dem Namen des Felds `$_POST` angeben. Die Eintragung im Texteingabefeld `vorname` wird also zum Wert der Variablen `$_POST["vorname"]` im Programm.

Die Absicherung von Programmen gegen böswillige Benutzer ist ein wichtiges Thema in PHP. In einem Texteingabefeld könnte schädlicher Code eingegeben werden, der nach der Übermittlung auf den Webserver zur Ausführung kommen könnte.

Daher wird die Funktion `htmlentities()` für den übermittelten Inhalt des Texteingabefelds aufgerufen. Sie wandelt alle HTML-spezifischen Zeichen in die entsprechenden *Entities* um. Im Quelltext der Seite, die von PHP erstellt wird, erscheinen dann zum Beispiel `<` für das Zeichen < und `>` für das Zeichen >. Normaler Textinhalt wird von dieser Umwandlung nicht beeinflusst. Zeichen, die schädlichen Code einleiten könnten, werden aber damit »entschärft«.

Diese Absicherung wird nur an denjenigen Stellen benötigt, an denen Benutzer Text eintragen können, der nicht in eine Zahl umgewandelt wird.

Übung »u_eingabe«, Teil 2

Erstellen Sie (passend zum Formular aus der Übung *u_eingabe*, Teil 1) ein PHP-Programm, das die Daten des Benutzers bestätigt (Datei *u_eingabe.php*). Hat der Benutzer die oben angegebenen Beispieldaten eingegeben, soll die Ausgabe des Programms im Browser so aussehen wie in Abbildung 1.16.

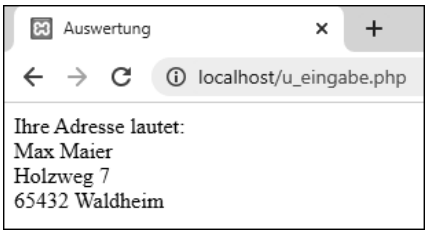


Abbildung 1.16 Auswertung des erweiterten Eingabeformulars

1.3.3 Umwandlungen zwischen Zeichenketten und Zahlen

Ein Texteingabefeld eines Formulars nimmt eine Zeichenkette auf; dabei wird eine Zeichenkette an das PHP-Programm übermittelt. Häufig werden die Eingaben jedoch als Zahlen benötigt, zum Beispiel zur Ausführung von Berechnungen.

Zeichenketten werden nach den folgenden Regeln implizit umgewandelt:

- Enthalten sie nach einem optionalen Vorzeichen nur Ziffern, werden sie in ganze Zahlen umgewandelt. Beispiele: `"42"`, `"-42"` oder `"+42"`.
- Enthalten sie zusätzlich einen Dezimalpunkt oder am Ende nach einem optionalen Vorzeichen und einem `e` oder `E` einen Exponenten, werden sie in Fließkommazahlen umgewandelt. Beispiele: `"4.2"`, `"-4.2"`, `"42e3"`, `"4.2e3"`, `"4.2e-3"` oder `"-4.2E3"`.

Andere Zeichen sollten in den Zeichenketten vermieden werden:

- Stehen andere Zeichen am Ende, wird nur der vordere Teil der Zeichenkette bis zum Beginn der anderen Zeichen umgewandelt. Allerdings erfolgt eine Warnung, dass es sich nicht um einen wohlgeformten numerischen Wert handelt. Beispiele: `"42abc23"` (Zahlenwert: 42) oder `"4.2 Liter"` (Zahlenwert 4.2).
- Stehen andere Zeichen am Beginn, ergibt sich der Zahlenwert 0 und es erfolgt ebenfalls die oben genannte Warnung. Beispiele: `"abc42"` oder `"Summe 4.2"`.

Sie können nicht wissen, was Benutzerinnen und Benutzer eingeben. Um die oben genannte Warnung zu vermeiden, sollten Sie daher Eingaben oder andere Zeichenketten, aus denen Sie Zahlenwerte entnehmen möchten, explizit in Zahlen umwandeln:

- Zur Umwandlung in eine ganze Zahl nutzen Sie die Funktion `intval()`. Dabei werden die Nachkommastellen abgeschnitten.
- Zur Umwandlung in eine Fließkommazahl nutzen Sie die Funktion `floatval()`. Diese Funktion können Sie auch über den Alias `doubleval()` aufrufen.

Dadurch, dass Sie die Eingabe der Benutzerinnen in eine Zahl umwandeln, verhindern Sie gleichzeitig die Übermittlung schädlichen Codes.

Umgekehrt können Sie Zahlen mithilfe der Funktion `strval()` explizit in eine Zeichenkette umwandeln. Das ist beim Aufruf von Funktionen sinnvoll, die nur mit einer Zeichenkette arbeiten können.

Ein Beispiel mit einigen Umwandlungen folgt:

```
<!DOCTYPE html>...<body>
<?php
   $a = 42;
   echo "Wert: $a<br>";
   echo "Als Fließkommazahl: " . floatval($a) . "<br>";
   $az = strval($a);
   echo "Als Text: $az, " . mb_strlen($az) . " Zeichen<br><br>";

   $a = 34.9;
   echo "Wert: $a<br>";
   echo "Als ganze Zahl: " . intval($a) . "<br>";
   $az = strval($a);
   echo "Als Text: $az, " . mb_strlen($az) . " Zeichen<br><br>";

   $a = "12.825";
   echo "Wert: $a, " . mb_strlen($a) . " Zeichen<br>";
   echo "Als ganze Zahl: " . intval($a) . "<br>";
   echo "Als Fließkommazahl: " . floatval($a);
?>
</body></html>
```

Listing 1.12 Die Datei »umwandeln.php«

Zunächst wird die ganze Zahl 42 in die Fließkommazahl 42.0 umgewandelt. Diese Zahl wird allerdings weiterhin als 42 ausgegeben. Anschließend wird die ganze Zahl 42 in die Zeichenkette `"42"` umgewandelt. Mithilfe der Zeichenkettenfunktion `mb_strlen()` wird ermittelt, dass diese Zeichenkette aus zwei Zeichen besteht. Mehr zu Zeichenkettenfunktionen finden Sie in Kapitel 6.

Als Nächstes wird die Fließkommazahl 34.9 in die ganze Zahl 34 umgewandelt. Dabei werden die Nachkommastellen abgeschnitten. Die Zeichenkette `"34.9"` besteht aus vier Zeichen.

Zuletzt wird die Zeichenkette `"12.825"`, die aus sechs Zeichen besteht, in die ganze Zahl 12 und in die Fließkommazahl 12.825 umgewandelt.

Die Ausgabe des Programms sehen Sie in Abbildung 1.17.

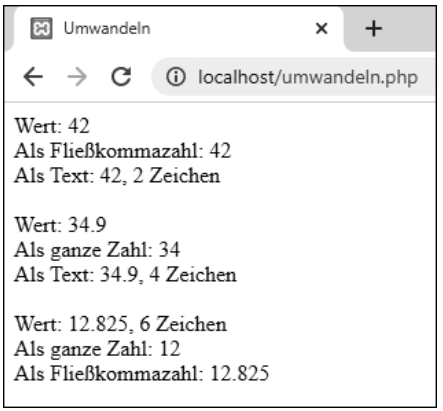


Abbildung 1.17 Umwandlungen zwischen Zeichenketten und Zahlen

[»] **Hinweis**

In den ersten Beispielen dieses Buches werden Eingabefehler des Benutzers nicht immer abgefangen. Die Programme würden sonst unnötig umfangreich und schwer verständlich. Später werden Routinen zum Abfangen von Eingabefehlern in die Programme eingebaut.

[»] **Hinweis**

Der Aufruf der Funktion `mb_strlen()` mit einer Zahl würde bei der Nutzung von Typhinweisen (siehe Abschnitt 1.10.11) zu einer Fehlermeldung führen.

Sollten Sie mit Ubuntu arbeiten und nicht meine empfohlene XAMPP-Installation nutzen, kann eine Fehlermeldung bei der Nutzung der Multibytefunktionen wie zum Bei-

spiel `mb_strlen()` erscheinen. In diesem Fall können Sie diese Funktionen mithilfe des folgenden Kommandozeilenbefehls nachinstallieren:

```
sudo apt-get install php-mbstring
```

1.3.4 Umwandlung von Eingaben

Im folgenden Beispiel werden die Benutzer aufgefordert, zwei Zahlen in ein Formular einzugeben und das Formular abzusenden. Bei den Eingaben handelt es sich um Zeichenketten. Diese werden mithilfe der Funktion `floatval()` in Fließkommazahlen umgewandelt. Ein Nebeneffekt davon: Der Einsatz der Funktion `htmlentities()` kann entfallen. Ein PHP-Programm berechnet die Summe der beiden Zahlen und gibt das Ergebnis aus. Der HTML-Code des Formulars lautet:

```
<!DOCTYPE html>...<body>
<p>Bitte zwei Zahlen eintragen und das Formular absenden</p>
<form action="eingabe_zahl.php" method="post">
    <p>Zahl 1: <input name="zahl1"></p>
    <p>Zahl 2: <input name="zahl2"></p>
    <p><input type="submit"> <input type="reset"></p>
</form>
</body></html>
```

Listing 1.13 Die Datei »eingabe_zahl.htm«

Das PHP-Programm sieht so aus:

```
<!DOCTYPE html>...<body>
<?php
   $zahl1 = floatval($_POST["zahl1"]);
   $zahl2 = floatval($_POST["zahl2"]);
   $summe = $zahl1 + $zahl2;
   echo "Die Summe von $zahl1 und $zahl2 ist $summe";
?>
</body></html>
```

Listing 1.14 Die Datei »eingabe_zahl.php«

Ein Aufruf mit den in Abbildung 1.18 dargestellten Eingabewerten ergibt die in Abbildung 1.19 gezeigte Antwort.

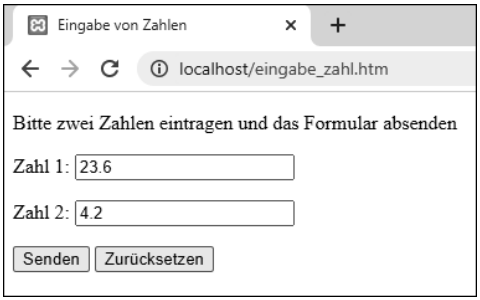


Abbildung 1.18 Senden von Zahlen

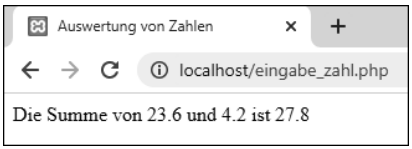


Abbildung 1.19 Umwandlung und Berechnung des Ergebnisses

[ ] **Übung »u_eingabe_zahl«**

Erstellen Sie ein Eingabeformular (Datei *u_eingabe_zahl.htm*) und ein dazu passendes PHP-Programm (Datei *u_eingabe_zahl.php*), mit dessen Hilfe das Quadrat einer Zahl berechnet werden kann. Die Zahl soll also mit sich selbst multipliziert werden. Vergessen Sie nicht, die eingegebene Zeichenkette in eine Zahl umzuwandeln.

Formular und Ergebnis sollten so wie in Abbildung 1.20 und Abbildung 1.21 aussehen.

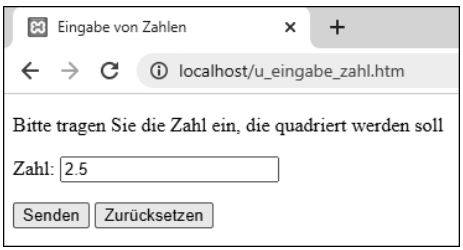


Abbildung 1.20 Eingabe der Übung »u_eingabe_zahl«

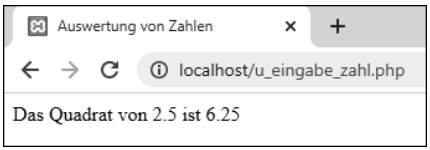


Abbildung 1.21 Ergebnis der Übung »u_eingabe_zahl«

1.4 Verzweigungen

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

Die Ausführung dieser Anweisungen wird in solchen Fällen von einer oder von mehreren Bedingungen abhängig gemacht. Je nachdem, ob die Bedingung zutrifft, werden die entsprechenden Anweisungen ausgeführt oder nicht. Darüber hinaus können bei Nichterfüllung der Bedingung alternative Anweisungen bearbeitet werden. Man nennt diese Stellen in einem Programm *Verzweigungen* oder auch *bedingte Anweisungen*.

Bedingungen werden mithilfe von Wahrheitswerten (wahr oder falsch) und Vergleichsoperatoren erstellt. Tabelle 1.2 enthält eine Übersicht über die Vergleichsoperatoren. Sie finden weitere Informationen über die Hintergründe von Wahrheitswerten in Abschnitt 1.5.1. Zunächst aber kommen wir zur praktischen Nutzung.

Operator	Bedeutung	Geltungsbereich
`==`	gleich	Zahlen und Zeichenketten
`!=`	ungleich	Zahlen und Zeichenketten
`>`	größer als	Zahlen
`<`	kleiner als	Zahlen
`>=`	größer als oder gleich	Zahlen
`<=`	kleiner als oder gleich	Zahlen

Tabelle 1.2 Vergleichsoperatoren in PHP

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

1.4.1 Einfache Verzweigung mit »if«

Hier sehen Sie ein Beispiel für eine einfache Verzweigung mit einer `if`-Anweisung:

```
<!DOCTYPE html>...<body>
<?php
   $x = 8;
   $y = 12;
```

```
   if($x < $y)
      echo "$x ist kleiner als $y";
?>
</body></html>
```

Listing 1.15 Die Datei »if.php«

Falls $x kleiner als $y ist, wird der entsprechende Text ausgegeben, andernfalls geschieht nichts. Die Bedingung (hier: $x < $y) muss in Klammern stehen. Die Ausgabe sehen Sie in Abbildung 1.22. Ändern Sie einmal kurzfristig den Wert von $x im PHP-Programm, zum Beispiel in 18. Es erfolgt keine Ausgabe mehr, da die Bedingung nicht erfüllt ist.

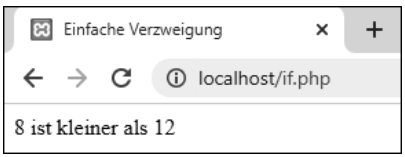


Abbildung 1.22 Einfache Verzweigung mit »if«

Ein weiteres Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   $x = 8;
   $y = 12;

   if($x < $y)
   {
      echo "$x ist kleiner als $y<br>";
      echo "$y ist größer als $x";
   }
?>
</body></html>
```

Listing 1.16 Die Datei »if_block.php«

Sollen aufgrund einer Bedingung mehrere Anweisungen ausgeführt werden, müssen diese innerhalb eines Anweisungsblocks stehen. Ein solcher Block wird mithilfe von geschweiften Klammern {} gebildet. Diese erreichen Sie auf der Tastatur mithilfe der Sondertaste Alt Gr.

In diesem Programm werden zwei Ausgaben erzeugt, da $x kleiner als $y ist. Abbildung 1.23 zeigt die Ausgabe.

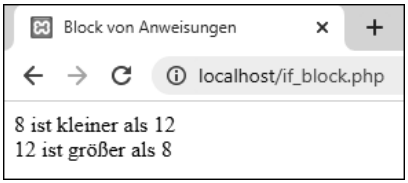


Abbildung 1.23 Verzweigung mit Anweisungsblock

1.4.2 Alternativer Zweig mit »else«

Bei Nichterfüllung einer Bedingung können gegebenenfalls alternative Anweisungen bearbeitet werden. Dazu wird zusätzlich das Schlüsselwort `else` benötigt:

```
<!DOCTYPE html>...<body>
<?php
   $x = 18;
   $y = 12;

   if($x < $y)
   {
      echo "$x ist kleiner als $y<br>";
      echo "$y ist größer als $x";
   }
   else
   {
      echo "$x ist größer oder gleich $y<br>";
      echo "$y ist kleiner oder gleich $x";
   }
?>
</body></html>
```

Listing 1.17 Die Datei »ifelse.php«

Trifft die Bedingung nach dem `if` nicht zu, wird die Anweisung oder der Anweisungsblock nach dem `else` ausgeführt. Die Ausgabe sehen Sie in Abbildung 1.24.

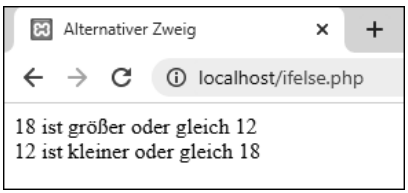


Abbildung 1.24 Verzweigung mit »if« und »else«

Ein weiteres Beispiel (mit Eingabeformular) verdeutlicht den Vergleich von Zeichenketten bei einer Bedingung. Dabei wird der Zugang zu einer Internetseite per Passwort (stark vereinfacht und ausnahmsweise in sichtbarer Form) simuliert. Das PHP-Programm vergleicht die Eingabe mit dem gespeicherten Passwort und reagiert entsprechend. Der HTML-Code des Formulars sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<p>Bitte tragen Sie das Zugangspasswort ein</p>
<form action="ifelse_zugang.php" method="post">
    <p><input name="passwort"></p>
    <p><input type="submit"> <input type="reset"></p>
</form>
</body></html>
```

Listing 1.18 Die Datei »ifelse_zugang.htm«

Das Auswertungsprogramm sieht so aus:

```
<!DOCTYPE html>...<body>
<?php
   $passwort = htmlentities($_POST["passwort"]);
   if($passwort == "bingo")
      echo "Zugang gestattet";
   else
      echo "Zugang verweigert";
?>
</body></html>
```

Listing 1.19 Die Datei »ifelse_zugang.php«

Gibt der Benutzer das Passwort aus Abbildung 1.25 ein ...

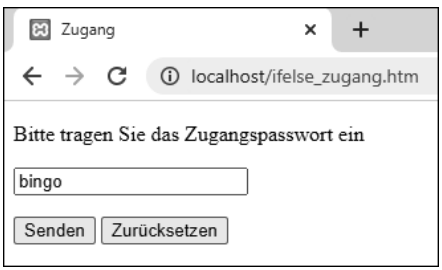


Abbildung 1.25 Eingabe des Passworts

... erhält er Zugang (siehe Abbildung 1.26), ...

Abbildung 1.26 Auswertung der Verzweigung

... andernfalls nicht.

Übung »u_ifelse1«

Erstellen Sie ein Eingabeformular (Datei *u_ifelse1.htm*) und ein dazu passendes PHP-Programm (Datei *u_ifelse1.php*). Es soll der Preis für eine Tankfüllung an einer Tankstelle berechnet werden. Es gibt zwei Sorten Benzin: Normal (Preis: 1,35 €) und Super (Preis: 1,40 €).

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

Gibt die Benutzerin also beispielsweise ein, dass sie 15,5 Liter Super-Benzin tankt (siehe Abbildung 1.27), ...

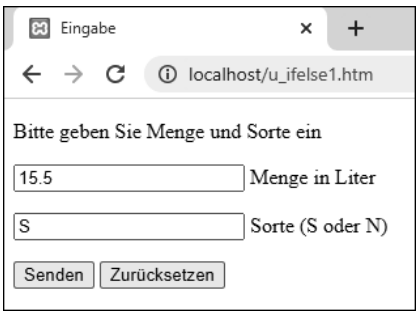


Abbildung 1.27 Eingabe des Tankvorgangs

... sollte die Ausgabe des Programms so aussehen wie in Abbildung 1.28.

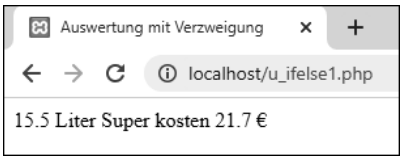


Abbildung 1.28 Ergebnis des Tankvorgangs

Übung »u_ifelse2«

Erweitern Sie die vorherige Übung. Großkunden, die 100 Liter oder mehr tanken, erhalten unabhängig von der Sorte an dieser Tankstelle 2 % Rabatt. Gibt ein Benutzer beispielsweise ein, dass er 120 Liter Normal-Benzin tankt (siehe Abbildung 1.29), ...

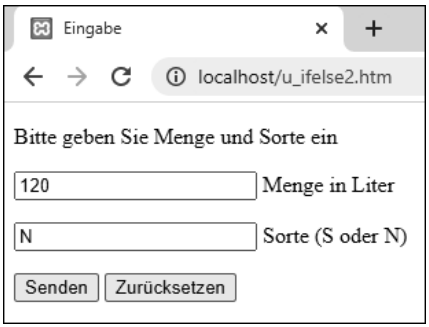


Abbildung 1.29 Eingabe der Übung »u_ifelse2«

... sollte die Ausgabe des Programms so aussehen wie in Abbildung 1.30.

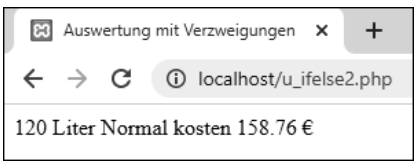


Abbildung 1.30 Ergebnis der Übung »u_ifelse2«

1.4.3 Verknüpfung mit »oder«

Logische Operatoren dienen zur Verknüpfung mehrerer Bedingungen. Zunächst wird das *logische Oder* vorgestellt. Es wird mithilfe der Zeichenfolge || gebildet. Es wird verwendet, falls nur eine von mehreren Bedingungen zutreffen muss. Das Zeichen | finden Sie auf einer Windows-Tastatur links unten bei den Kleiner/Größer-Zeichen. Sie müssen zusätzlich die Sondertaste Alt Gr betätigen.

Zur Verdeutlichung erweitern wir nun das Beispiel mit der Passworteingabe, das Sie in den Dateien *ifelse_zugang.htm* und *ifelse_zugang.php* finden. Es gibt nun zwei Passwörter, die zum erfolgreichen Zugang führen. Das Eingabeformular in der Datei *oder.htm* entspricht demjenigen in der Datei *ifelse_zugang.htm*; das Auswertungsprogramm sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   $passwort = htmlentities($_POST["passwort"]);
   if($passwort == "bingo" || $passwort == "kuckuck")
      echo "Zugang gestattet";
   else
      echo "Zugang verweigert";
?>
</body></html>
```

Listing 1.20 Die Datei »oder.php«

Mindestens eine der beiden Bedingungen muss zutreffen, damit der Zugang gestattet wird. Jede Bedingung muss vollständig formuliert werden. Der Ausdruck `$passwort == "bingo" || "kuckuck"` würde zu einer Fehlermeldung führen, da die zweite Bedingung unvollständig ist.

Die Vergleichsoperatoren, die zur Auswertung der Bedingungen benötigt werden, stehen in der Rangordnung der Operatoren höher als der logische Operator `||`. Daher werden zunächst die einzelnen Bedingungen geprüft. Erst anschließend wird der logische Operator zur Verknüpfung der Ergebnisse der Bedingungen eingesetzt. Daher ist es nicht notwendig, die einzelnen Bedingungen in Klammern zu setzen.

1.4.4 Verknüpfung mit »und«

Das *logische Und* (Zeichenfolge `&&`) wird verwendet, wenn alle Bedingungen zutreffen müssen. Dies wird wiederum an einem erweiterten Beispiel der Passworteingabe verdeutlicht. Der Benutzer muss nun seinen Namen und sein Zugangspasswort eingeben. Der Zugang wird nur gestattet, wenn beide Angaben korrekt sind, es sich also um einen sowohl berechtigten als auch bekannten Benutzer handelt. Hier sehen Sie zunächst das geänderte Eingabeformular:

```
<!DOCTYPE html>...<body>
<p>Bitte tragen Sie Name und Zugangspasswort ein</p>
<form action="und.php" method="post">
    <p><input name="benutzer"> Name des Benutzers</p>
    <p><input name="passwort"> Passwort</p>
    <p><input type="submit"> <input type="reset"></p>
</form>
</body></html>
```

Listing 1.21 Die Datei »und.htm«

Das Auswertungsprogramm sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   $benutzer = htmlentities($_POST["benutzer"]);
   $passwort = htmlentities($_POST["passwort"]);
   if($benutzer == "Maier" && $passwort == "kuckuck")
      echo "Zugang gestattet";
   else
      echo "Zugang verweigert";
?>
</body></html>
```

Listing 1.22 Die Datei »und.php«

Gibt der Benutzer zwar den Namen `Maier`, aber ein falsches Passwort ein, wird der Zugang verweigert, da beide Angaben stimmen müssen. Das Gleiche trifft zu, wenn der Benutzer den Namen `Meier` (mit `e` statt mit `a`) und das richtige Passwort `kuckuck` eingibt, da in diesem Fall nur die zweite Bedingung zutrifft (siehe das Formular in Abbildung 1.31 und die Ausgabe in Abbildung 1.32).

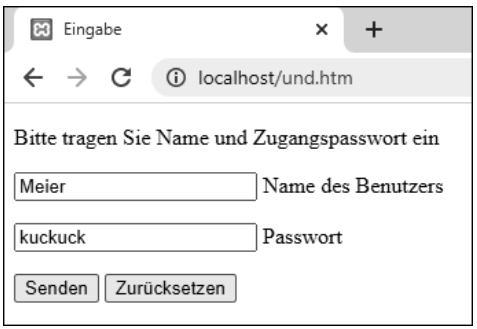


Abbildung 1.31 Eingabe des Namens und des Passworts

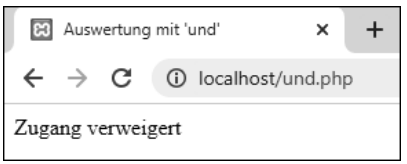


Abbildung 1.32 Richtiges Passwort, falscher Name

Auch hier gilt, dass die Vergleichsoperatoren in der Rangordnung der Operatoren höher stehen als der logische Operator (hier: &&) und daher die Bedingungen nicht in Klammern gesetzt werden müssen.

> **Übung »u_oder_und«** [ ]
>
> Testen Sie die Beispiele in den Dateien *oder.htm* und *oder.php* beziehungsweise *und.htm* und *und.php* mit verschiedenen Passwörtern sowie Name-Passwort-Kombinationen.

1.4.5 Umkehrung mit »nicht«

Mithilfe des *logischen Nicht* (Operator !) wird der Wahrheitswert einer Bedingung umgekehrt. Damit ist es möglich, eine Bedingung oder eine logische Verknüpfung anders zu formulieren und sie damit eventuell lesbarer zu machen. Ein Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   $zahl = 16;

   if($zahl < 20)
      echo "Zahl ist kleiner als 20<br>";
   if(!($zahl >= 20))
      echo "Zahl ist kleiner als 20<br>";

   if($zahl < 20 || $zahl > 30)
      echo "Zahl liegt nicht zwischen 20 und 30<br>";
   if(!($zahl >= 20 && $zahl <= 30))
      echo "Zahl liegt nicht zwischen 20 und 30<br>";
?>
</body></html>
```

Listing 1.23 Die Datei »nicht.php«

In beiden Fällen liefern beide Verzweigungen dasselbe Ergebnis. Der Operator ! steht in der Rangordnung höher als die Vergleichsoperatoren. Daher muss die Bedingung beziehungsweise die logische Verknüpfung in Klammern gesetzt werden. Die Ausgabe sehen Sie in Abbildung 1.33.

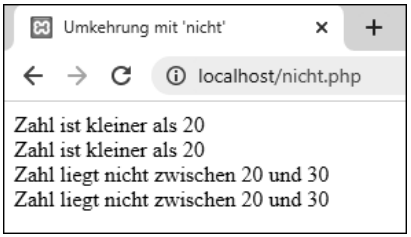


Abbildung 1.33 Umkehrung des Wahrheitswerts

1.4.6 Rangordnung der Operatoren

Ausdrücke mit mehreren Operatoren werden von links nach rechts aufgelöst – unter Beachtung der Rangordnung. In Tabelle 1.3 sehen Sie die Rangordnung der bisher verwendeten Operatoren. Die Tabelle beginnt mit der höchsten Stelle der Rangordnung.

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

Tabelle 1.3 Rangordnung der Operatoren

Klammern stehen in der Rangordnung an erster Stelle. Mit ihrer Hilfe können Sie Ausdrücke in der gewünschten Reihenfolge bearbeiten lassen.

Übung »u_logisch«

Erweitern Sie das Beispielprogramm aus dem vorherigen Abschnitt. Nur die beiden Benutzer `Marten` (Passwort `Hamburg`) und `Schmitz` (Passwort `Berlin`) sollen Zugang haben. Verwenden Sie die Dateien *u_logisch.htm* und *u_logisch.php*.

1.4.7 Mehrfache Verzweigung mit »if« und »else«

Verzweigungen mit `if` und `else` lassen sich verschachteln, sodass eine mehrfache Verzweigung möglich wird. Eine solche Verzweigung kann für drei oder noch mehr Fälle verwendet werden. Ein Beispiel hierzu:

```
<!DOCTYPE html>...<body>
<?php
   $x = 8;
   $y = 12;

   if($x < $y)
   {
      echo "$x ist kleiner als $y<br>";
      echo "$y ist größer als $x";
   }
   else
   {
      if($x > $y)
      {
         echo "$x ist größer als $y<br>";
         echo "$y ist kleiner als $x";
      }
      else
         echo "Beide sind gleich";
   }
?>
</body></html>
```

Listing 1.24 Die Datei »if_mehrfach.php«

Falls `$x` kleiner als `$y` ist, trifft die erste Bedingung zu. Die restlichen Bedingungen müssen in diesem Fall nicht mehr geprüft werden. Andernfalls kann `$x` nur noch größer oder gleich `$y` sein, und es wird die nächste Bedingung (`$x > $y`) geprüft. Trifft diese ebenfalls nicht zu, kann `$x` nur noch gleich `$y` sein. Die Ausgabe für den letztgenannten Fall sehen Sie in Abbildung 1.34.

Abbildung 1.34 Ergebnis mehrfacher Verzweigung

Übung »u_if_mehrfach«

Erweitern Sie das Programm aus der Übung *u_ifelse1*. Nun soll der Preis für eine Tankfüllung ohne Rabatt für Großkunden berechnet werden. Es gibt drei Sorten Benzin: Normal (Preis: 1,35 €), Super (Preis: 1,40 €) und Diesel (Preis: 1,10 €).

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

Tankt der Benutzer 35 Liter Diesel (siehe Abbildung 1.35), ...

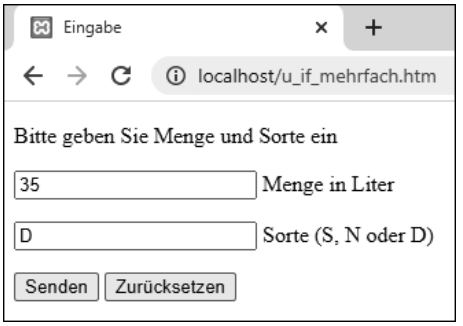


Abbildung 1.35 Eingabe der Übung »u_if_mehrfach«

... sollte die Ausgabe so wie in Abbildung 1.36 aussehen.

Abbildung 1.36 Ergebnis der Übung »u_if_mehrfach«

1.4.8 Mehrfache Verzweigung mit »switch«

Die `switch`-Anweisung bietet eine alternative Schreibweise für einen bestimmten Typ von mehrfachen Verzweigungen. Sie kann eingesetzt werden, wenn eine Variable mit mehreren bestimmten Werten verglichen werden soll. Diese Form der mehrfachen Verzweigung kann übersichtlicher sein als eine verschachtelte Verzweigung, falls viele unterschiedliche Fälle vorliegen.

Es folgt ein Programm, das aus zwei Teilen besteht:

```
<!DOCTYPE html>...<body>
<?php
   $wuerfel = 3;

   /* Einzelne Fälle */
   switch($wuerfel)
   {
      case 1:
         $ausgabe = "Eins";
         break;
      case 2:
         $ausgabe = "Zwei";
         break;
      case 3:
         $ausgabe = "Drei";
         break;
      case 4:
         $ausgabe = "Vier";
         break;
      case 5:
         $ausgabe = "Fünf";
         break;
      case 6:
         $ausgabe = "Sechs";
         break;
      default:
         $ausgabe = "Kein Würfelwert";
   }
   echo $ausgabe . "<br>";

   /* Zusammengefasste Fälle */
   switch($wuerfel)
   {
      case 1:
      case 3:
      case 5:
         $ausgabe = "Ungerade Zahl";
         break;
      case 2:
```

```
        case 4:
        case 6:
            $ausgabe = "Gerade Zahl";
            break;
        default:
            $ausgabe = "Kein Würfelwert";
    }
    echo $ausgabe . "<br>";
?>
</body></html>
```

Listing 1.25 Die Datei »switch.php«

Nehmen wir an, die Variable `$wuerfel` enthält einen zufällig ermittelten Würfelwert.

Im ersten Teil des Programms wird der Wert der Variablen mithilfe eines `switch`-Blocks untersucht. Die einzelnen vorhandenen Fälle (englisch: *cases*) werden der Reihe nach durchlaufen. Sobald einer der Fälle zutrifft, die Variable also den Wert nach dem Schlüsselwort `case` besitzt, werden die zugehörigen Anweisungen bis zum nächsten Auftreten des Schlüsselworts `break` bearbeitet. Anschließend wird der `switch`-Block unmittelbar verlassen.

Optional kann es einen `default`-Fall geben. Er wird verwendet, falls keiner der genannten Fälle zutrifft, die Variable im vorliegenden Beispiel also keinen Würfelwert enthält.

Im zweiten Teil des Programms werden einige Fälle zusammengefasst. Beim Würfelwert 2 und beim Würfelwert 6 passiert dasselbe, da das nächste `break` erst anschließend folgt.

Die Ausgabe des Programms sehen Sie in Abbildung 1.37.

Abbildung 1.37 Mehrfache Verzweigung mit »switch«

1.4.9 Mehrfache Verzweigung mit »match«

Mit PHP 8.0 wurde der `match`-Ausdruck eingeführt. Er bietet eine weitere Möglichkeit für eine mehrfache Verzweigung. Zum Vergleich folgt ein Programm, das denselben Ablauf wie das Programm aus dem vorherigen Abschnitt besitzt:

```
<!DOCTYPE html>...<body>
<?php
   $wuerfel = 3;

   /* Einzelne Fälle */
   echo match($wuerfel)
   {
      1 => "Eins",
      2 => "Zwei",
      3 => "Drei",
      4 => "Vier",
      5 => "Fünf",
      6 => "Sechs",
      default => "Kein Würfelwert"
   };
   echo "<br>";

   /* Zusammengefasste Fälle */
   echo match($wuerfel)
   {
      1, 3, 5 => "Ungerade Zahl",
      2, 4, 6 => "Gerade Zahl",
      default => "Kein Würfelwert"
   };
   echo "<br>";
?>
</body></html>
```

Listing 1.26 Die Datei »match.php«

Ein `match`-Ausdruck liefert ein Ergebnis zurück, ähnlich wie eine Funktion. Dieses Ergebnis kann gespeichert oder mithilfe von `echo` ausgegeben werden. Die verschiedenen Fälle werden ohne das Schlüsselwort `case` gebildet. Nach dem Operator `=>` folgt ein Wert als Wert des gesamten `match`-Ausdrucks, gefolgt von einem Komma.

Es sollte immer einen `default`-Fall geben. Trifft keiner der vorherigen Fälle zu, wird dieser Fall genutzt. Gäbe es keinen `default`-Fall, würde ein Fehler auftreten. Nach der schließenden Klammer des `match`-Ausdrucks folgt ein Semikolon.

Auch in einem `match`-Ausdruck können mehrere Fälle zusammengefasst werden. Sie werden durch Kommata voneinander getrennt. Die Ausgabe sehen Sie in Abbildung 1.38.

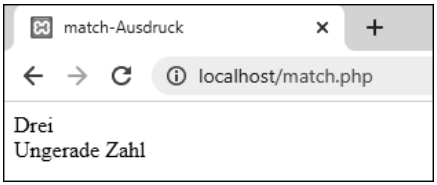


Abbildung 1.38 Mehrfache Verzweigung mit »match«

Weitere Unterschiede zwischen `switch` und `match` sind:

- In einem `switch`-Block können mehrere Anweisungen ausgeführt werden. In einem `match`-Ausdruck wird nur ein einzelner Wert zugewiesen.
- In einem `switch`-Block werden nur die Werte verglichen. In einem `match`-Ausdruck werden sowohl die Werte als auch die Datentypen verglichen (siehe auch Abschnitt 1.5.1).
- Ein `match`-Ausdruck enthält automatisch ein `break`. Es ist also nicht nötig, `break` einzugeben wie bei der Nutzung von `switch` (vgl. Listing 1.25).

1.5 Mehr über Verzweigungen

Nachdem Sie die Grundlagen zum Thema »Verzweigungen« kennengelernt haben, erläutere ich in diesem Abschnitt einige weitergehende Möglichkeiten. Sie können diesen Abschnitt auch zunächst überspringen und unmittelbar mit Abschnitt 1.6 über das Thema »Schleifen« fortfahren.

1.5.1 Wahrheitswerte

In diesem Abschnitt wird das Wissen über Wahrheitswerte vertieft, die zum Beispiel innerhalb von Bedingungen benötigt werden. Diese Wahrheitswerte können in eigenen Variablen zwischengespeichert werden, um sie später zu nutzen. Dazu dient der Datentyp `boolean`. In den Variablen dieses Datentyps wird entweder `true` (wahr) oder `false` (falsch) gespeichert. Sie werden auch *boolesche Variablen* genannt.

Zahlen, Zeichenketten und Variablen besitzen ebenfalls einen Wahrheitswert, den Sie in Ihren Programmen nutzen können. Diese Nutzung kann implizit erfolgen, also durch eine automatische Umwandlung; sie kann aber auch explizit mithilfe der Funktion `boolval()` erfolgen.

Mithilfe der »strengen« Vergleichsoperatoren `===` und `!==` ermitteln Sie, ob zwei Werte übereinstimmen *und* denselben Datentyp haben.

Hier folgen einige Wahrheitswerte, Umwandlungen und Vergleiche:

```
<!DOCTYPE html>...<body>
<?php
   $ww = 5>3;
   echo "Wahrheitswert: $ww<br>";
   if($ww) echo "Dieser Wert ist wahr<br><br>";

   echo "Implizit: 5>3: " . (5>3) . ", 5<3: " . (5<3) . "<br>";
   echo "Explizit: boolval(5>3): " . boolval(5>3) .
      ", boolval(5<3): " . boolval(5<3) . "<br><br>";

   echo "TRUE: " . TRUE . ", true: " . true . "<br>";
   echo "FALSE: " . FALSE . ", false: " . false . "<br><br>";

   echo "boolval(1): " . boolval(1) . ", boolval(0): " . boolval(0)
      . ", boolval(-1): " . boolval(-1) . "<br>";
   echo "boolval(0.0): " . boolval(0.0) . ", boolval(0.000000001): "
      . boolval(0.000000001) . "<br>";
   echo "boolval(''): " . boolval('') . ", boolval(' '): " . boolval(' ')
      . ", boolval('0'): " . boolval('0') . "<br><br>";

   $zahl = 42;
   $text = "42";
   if($zahl == $text) echo "==<br>";
   if($zahl != $text) echo "!=<br>";
   if($zahl === $text) echo "===<br>";
   if($zahl !== $text) echo "!==<br>";
?>
</body></html>
```

Listing 1.27 Die Datei »wahrheitswert.php«

In der Variablen `$ww` wird der Wahrheitswert einer Bedingung gespeichert und ausgegeben. Der Wahrheitswert `true` erscheint als `1`. Er kann innerhalb einer Verzweigung genutzt werden, zum Beispiel anstelle einer Bedingung oder verknüpft mit einer weiteren Bedingung.

Es wird sowohl der Wahrheitswert einer wahren als auch einer falschen Bedingung direkt ausgegeben, einmal nach impliziter Umwandlung, einmal nach expliziter Um-

wandlung mithilfe von `boolval()`. Für den Wahrheitswert `false` wird kein sichtbares Zeichen ausgegeben.

Sie können die Wahrheitswerte `true` und `false` auch direkt zuweisen. Dabei ist es egal, ob Sie Groß- oder Kleinschreibung anwenden.

Die Zahlenwerte `0` und `0.0`, die leere Zeichenkette und die Zeichenkette `"0"` beziehungsweise `'0'` entsprechen dem Wahrheitswert `false`. Alle anderen Zahlen und Zeichenketten entsprechen `true`.

Bei einem Vergleich mit einem der beiden Operatoren `==` oder `!=` ist es nicht wichtig, ob die beiden Werte denselben Datentyp besitzen. Die Zahl 42 entspricht also einer Zeichenkette, die nach Umwandlung den Zahlenwert 42 liefert, also zum Beispiel `"42"` oder `"42abc"`.

Ist jedoch für den Wahrheitswert einer Bedingung auch der Datentyp entscheidend, müssen Sie einen der beiden strengen Vergleichsoperatoren `===` oder `!==` verwenden.

Es gibt Funktionen, die im Fehlerfall den Wahrheitswert `false` zurückliefern. Ein Teil dieser Funktionen kann im Erfolgsfall einen Wert zurückliefern, der ebenfalls als `false` ausgewertet werden kann. Den Erfolg des Aufrufs einer solchen Funktion müssen Sie daher ebenfalls mit einem strengen Vergleichsoperator prüfen.

Die Ausgabe des Programms sehen Sie in Abbildung 1.39.

Abbildung 1.39 Wahrheitswerte, Umwandlungen und Vergleiche

1.5.2 Ternärer Operator ?:

Der *ternäre Operator* `?:` kann in vielen Fällen als kompakte Abkürzung einer Verzweigung dienen. Er kombiniert eine Verzweigung mit einer Zuweisung, ähnlich wie der `match`-Ausdruck.

Ein Beispiel sehen Sie im folgenden Programm:

```
<!DOCTYPE html>...<body>
<?php
   $x = 8;
   $y = 12;

   $z = $x < $y ? $x : $y;
   echo "Kleinere Zahl: $z<br>";

   echo "Kleinere Zahl: " . ($x < $y ? $x : $y) . "<br>";

   $f = $x < $y ? "1" : ($x > $y ? "2" : "3");
   echo "Fall $f<br>";
?>
</body></html>
```

Listing 1.28 Die Datei »ternaer.php«

Viele Operatoren arbeiten mit zwei Operanden. Das gilt zum Beispiel für den Additionsoperator `+` im Ausdruck `3+5`. Vor und hinter dem Operator stehen die beiden Operanden, die addiert werden, hier `3` und `5`.

Ein ternärer Operator arbeitet dagegen immer mit drei Operanden. Beim ternären Operator `?:` steht vor dem Zeichen `?` eine Bedingung (hier `$x < $y`). Trifft sie zu (wie beim `if`), wird als Ergebnis der Wert geliefert, der zwischen dem Zeichen `?` und dem Zeichen `:` steht. Trifft sie nicht zu (wie beim `else`), wird als Ergebnis der Wert geliefert, der nach dem Zeichen `:` steht.

Beim ersten Einsatz des ternären Operators wird das Ergebnis in der Variablen `$z` gespeichert. Wird das Ergebnis des ternären Operators direkt ausgegeben, wie beim zweiten Einsatz, sollte die gesamte Verzweigung in Klammern stehen. Ansonsten wird die Anweisung in der falschen Reihenfolge bearbeitet.

Wie bei einer mehrfachen Verzweigung mit `if` kann der ternäre Operator ebenfalls für drei oder mehr Fälle genutzt werden. Der Variablen `$f` wird eines von drei möglichen Ergebnissen zugewiesen. Ist `$x` kleiner als `$y`, lautet das Ergebnis `"1"`. Ansonsten wird mit-

hilfe eines weiteren ternären Operators geprüft, ob `$x` größer als `$y` ist. In diesem Fall ist das Ergebnis `"2"`, ansonsten `"3"`.

Die Ausgabe sehen Sie in Abbildung 1.40.

Abbildung 1.40 Verzweigungen mit dem ternären Operator »?:«

1.5.3 Spaceship-Operator <=>

Der *Spaceship-Operator* `<=>` wurde mit PHP 7.0 eingeführt. Er zählt zu den Vergleichsoperatoren und steht in der Rangordnung der Operatoren auf gleicher Höhe mit den Operatoren `==` und `!=`.

Ein Vergleich mit dem Operator `<=>` ergibt

- den Wert `1`, falls der erste Wert größer ist,
- oder den Wert `-1`, falls der zweite Wert größer ist,
- oder den Wert `0`, falls beide Werte übereinstimmen.

Ein Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   echo "Erster Wert: " . (12 <=> 5) . "<br>";
   echo "Zweiter Wert: " . (5 <=> 12) . "<br>";
   echo "Werte sind gleich: " . (5 <=> 5)  . "<br>";
?>
</body></html>
```

Listing 1.29 Die Datei »spaceship.php«

Die Ausgabe sehen Sie in Abbildung 1.41. Würden Sie die runden Klammern um den Vergleich mit dem Spaceship-Operator weglassen, würden die beiden Zeichenketten davor beziehungsweise dahinter in den Vergleich mit einbezogen. Das führt zu falschen Ergebnissen.

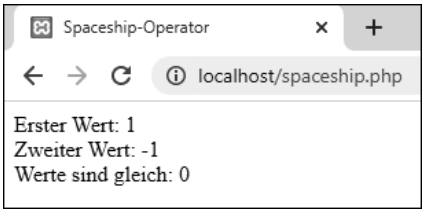


Abbildung 1.41 Auswertung mithilfe des Spaceship-Operators <=>

Der Spaceship-Operator trägt seinen Namen in Anlehnung an frühere textbasierte Computerspiele, in denen ein Raumschiff mithilfe der Zeichenfolge `<=>` angezeigt wurde.

1.5.4 Existenz von Variablen

Sie können die Existenz einer Variablen mithilfe der Funktion `isset()` prüfen. Auf diese Weise können Sie zum Beispiel feststellen, ob ein Kontrollkästchen in einem Formular vor dem Absenden markiert wurde oder nicht (siehe Abschnitt 2.2.3). Die Funktion `isset()` liefert einen Wahrheitswert, daher wird sie meist innerhalb einer Verzweigung eingesetzt und auch deshalb an dieser Stelle erläutert.

In engem Zusammenhang mit der Funktion `isset()` steht die Funktion `unset()`. Sie dient zum Löschen einer Variablen. Eine Variable kann ebenso gelöscht werden, indem man ihr den Wert `null` zuweist. Nachfolgend ein Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   if(isset($x)) echo "1: $x<br>";
   else          echo "1: Nicht vorhanden<br>";

   $x = 42;
   if(isset($x)) echo "2: $x<br>";
   else          echo "2: Nicht vorhanden<br>";

   unset($x);
   if(isset($x)) echo "3: $x<br>";
   else          echo "3: Nicht vorhanden<br>";

   $x = 42;
   if(isset($x)) echo "4: $x<br>";
   else          echo "4: Nicht vorhanden<br>";
```

```
    $x = null;
    if(isset($x)) echo "5: $x<br>";
    else          echo "5: Nicht vorhanden<br>";
?>
</body></html>
```

Listing 1.30 Die Datei »existenz.php«

Die Existenz der Variablen `$x` wird innerhalb des Programms mehrfach mithilfe der Funktion `isset()` geprüft. Die Verzweigung wird hier etwas kompakter innerhalb von zwei statt vier Zeilen notiert. Wenn die Variable `$x` existiert, wird ihr Wert ausgegeben. Wenn sie noch nicht existiert oder gelöscht wurde, wird eine entsprechende Information ausgegeben.

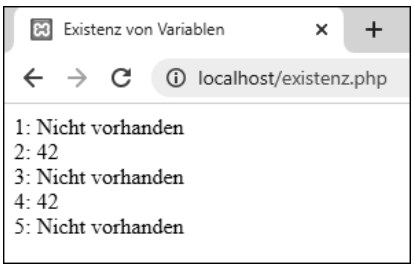


Abbildung 1.42 Die Existenz einer Variablen prüfen

Die Ausgabe des Programms inklusive Nummerierung sehen Sie in Abbildung 1.42. Zu den einzelnen nummerierten Ausgaben:

1. Vor der Zuweisung eines Werts existiert die Variable nicht.
2. Nach der Zuweisung eines Werts existiert die Variable.
3. Nach der Löschung mithilfe von `unset()` existiert sie nicht mehr.
4. Nach erneuter Zuweisung existiert sie wieder.
5. Nach der Löschung durch die Zuweisung von `null` existiert sie nicht mehr.

[»] **Hinweis**

Viele vordefinierte Funktionen liefern entweder den ermittelten Rückgabewert zurück oder den Wert `null` als Information dafür, dass innerhalb der Funktion ein Fehler aufgetreten ist.

1.5.5 Typ prüfen

Es gibt eine Reihe von Prüffunktionen, mit deren Hilfe Sie den Typ einer Variablen oder eines Werts feststellen können:

- `is_int()` prüft, ob es sich um eine ganze Zahl handelt.
- `is_float()` prüft, ob es sich um eine Fließkommazahl handelt.
- `is_string()` prüft, ob es sich um eine Zeichenkette handelt.
- `is_numeric()` prüft, ob es sich um eine Zahl handelt.
- `is_bool()` prüft, ob es sich um einen Wahrheitswert handelt.

Nachfolgend sehen Sie ein Programm mit einigen typischen Prüfungen:

```
<!DOCTYPE html>...<body>
<?php
   if(is_int(42))            echo "42 ist eine ganze Zahl<br>";
   if(!is_int(42.0))         echo "42.0 ist keine ganze Zahl<br>";
   if(is_float(42.0))        echo "42.0 ist eine Fliesskommazahl<br>";
   if(!is_float(42))         echo "42 ist keine Fliesskommazahl<br><br>";

   if(is_string("42"))       echo "\"42\" ist eine Zeichenkette<br>";
   if(is_string('42'))       echo "'42' ist eine Zeichenkette<br>";
   if(!is_string(42))        echo "42 ist keine Zeichenkette<br><br>";

   if(is_numeric("42"))      echo "\"42\" ist numerisch<br>";
   if(is_numeric("42.0"))    echo "\"42.0\" ist numerisch<br>";
   if(is_numeric("-4.2e-3")) echo "\"-4.2e-3\" ist numerisch<br>";
   if(!is_numeric("42a"))    echo "\"42a\" ist nicht numerisch<br><br>";

   if(is_bool(true))         echo "true ist boolean<br>";
   if(is_bool(5>3 && 7<12))  echo "5>3 && 7<12 ist boolean<br>";
   if(!is_bool("true"))      echo "\"true\" ist nicht boolean<br><br>";
?>
</body></html>
```

Listing 1.31 Die Datei »typ_pruefen.php«

In Abbildung 1.43 sehen Sie die Ausgabe des Programms.

Sobald eine Zahl einen Dezimalpunkt mit einer Nachkommastelle hat, wird sie von der Funktion `is_int()` nicht mehr als ganze Zahl erkannt. Sie wird aber von der Funktion `is_float()` als Fließkommazahl erkannt, selbst wenn die Nachkommastelle den Wert 0 hat.

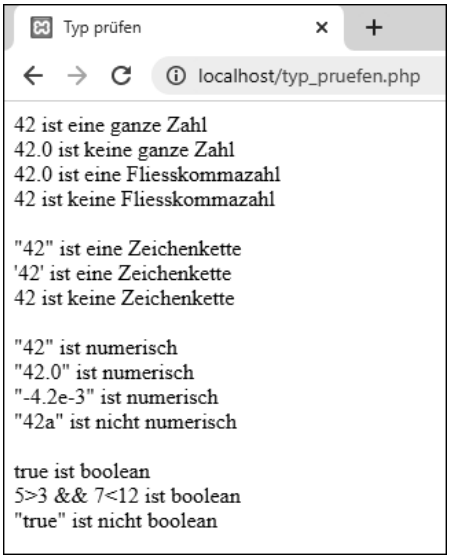


Abbildung 1.43 Das Ergebnis einiger Typprüfungen

Alles innerhalb von einfachen oder doppelten Hochkommata wird von der Funktion `is_string()` als Zeichenkette erkannt.

Ganze Zahlen, Fließkommazahlen oder Exponentialzahlen werden auch innerhalb einer Zeichenkette von der Funktion `is_numeric()` als gültige Zahlenwerte erkannt. Ebenso trifft das zu, wenn sie ein negatives Vorzeichen oder einen negativen Exponenten besitzen. Sobald innerhalb der Zeichenkette ein ungültiges Zeichen vorkommt, werden sie nicht mehr als gültige Zahlenwerte erkannt.

Die Werte `true` und `false` werden von der Funktion `is_bool()` als Wahrheitswerte erkannt. Das trifft auch für Bedingungen zu, die mithilfe von Vergleichsoperatoren und logischen Operatoren gebildet werden. Inhalte von Zeichenketten werden nicht als Wahrheitswerte erkannt.

Weitere Funktionen zur Typprüfung finden Sie in Abschnitt 4.9 und in Abschnitt 8.6.

1.5.6 Die Koaleszenzoperatoren ?? und ??=

Der Koaleszenzoperator `??` wurde mit PHP 7.0 eingeführt. Er verschmilzt die Arbeitsweise der Funktion `isset()` mit der Arbeitsweise des ternären Operators `?:` und wird daher auch *isset ternary*-Operator genannt. Damit kann die Ermittlung eines Werts verkürzt werden.

Seit PHP 7.4 gibt es auch den kombinierten Zuweisungsoperator `??=`. Er verbindet den Koaleszenzoperator mit einer Zuweisung. Damit wird die Ermittlung des Werts, der zugewiesen werden soll, verkürzt.

Einige Beispiele:

```
<!DOCTYPE html>...<body>
<?php
   echo $temperatur ?? "Temperatur nicht vorhanden";
   echo "<br>";
   $temperatur = 36.5;
   echo $temperatur ?? "Temperatur nicht vorhanden";
   echo "<br>";

   $luftdruck ??= "Luftdruck nicht vorhanden";
   echo $luftdruck;
   echo "<br>";
   $luftdruck = 1013.25;
   $luftdruck ??= "Luftdruck nicht vorhanden";
   echo $luftdruck;
?>
</body></html>
```

Listing 1.32 Die Datei »koaleszenz.php«

Existiert eine Variable, wird ihr Wert ausgegeben beziehungsweise zugewiesen. Existiert sie noch nicht oder wurde sie bereits wieder gelöscht, wird eine entsprechende Information ausgegeben beziehungsweise zugewiesen.

Die Ausgabe des Programms sehen Sie in Abbildung 1.44.

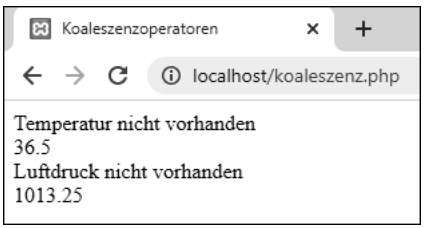


Abbildung 1.44 Koaleszenzoperatoren

Hinweis

Die Koaleszenzoperatoren tragen ihren Namen in Anlehnung an die Verschmelzung (englisch: *coalescence*) unterschiedlicher Flüssigkeiten.

[«]

1.6 Schleifen

Wiederholen sich innerhalb eines Programms einzelne Anweisungen oder Blöcke von Anweisungen, sollten Schleifen verwendet werden. In PHP gibt es unter anderem die `for`-Schleife, die `while`-Schleife und die `do-while`-Schleife. Welche Variante bei der Lösung eines aktuellen Problems die richtige ist, lässt sich leicht entscheiden:

- Sie verwenden die `for`-Schleife, wenn Ihnen die Anzahl der Wiederholungen bekannt ist oder sich diese bereits eindeutig vor Beginn der Schleife ergeben hat.
- Sie verwenden die `while`-Schleife oder die `do-while`-Schleife, wenn Ihnen die Anzahl der Wiederholungen nicht bekannt ist beziehungsweise wenn sich der Ablauf der Schleife erst zur Laufzeit des Programms ergibt (bedingungsgesteuerte Schleife).

1.6.1 Schleife mit »for«

Die `for`-Schleife wird verwendet, um 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:

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

Listing 1.33 Die Datei »for.php«

Mithilfe des Programms werden fünf Zeilen in das Dokument geschrieben, jeweils mit dem Inhalt `Zeile: <Nummer>`. Die Ausgabe sehen Sie in Abbildung 1.45.

Abbildung 1.45 Schleife mit »for«

Die `for`-Schleife besteht aus Kopf und Rumpf. Der Kopf der `for`-Schleife steht in Klammern. Beachten Sie, dass nach den Klammern kein Semikolon folgt. Innerhalb der Klammern stehen drei Elemente, die durch Semikola voneinander getrennt sind:

- Startwert
- Bedingung zur Wiederholung
- Veränderung der Schleifenvariablen

In diesem Beispiel wird die Variable `$i` als sogenannte *Schleifenvariable* verwendet. Das heißt, mithilfe von `$i` wird die Schleife gesteuert.

Die Variable `$i` bekommt zunächst den Wert `1`. Es wird geprüft, ob die Bedingung zum Durchlaufen der Schleife erfüllt ist. Ist das der Fall, wird mit dem Anfangswert der Rumpf der Schleife durchlaufen. Dies liefert die Ausgabe `Zeile 1`. Danach wird die Variable `$i` mithilfe des Inkrement-Operators `++` (siehe auch Abschnitt 1.2.5) auf `2` erhöht.

Anschließend wird geprüft, ob die Bedingung zur Wiederholung zum erneuten Durchlaufen der Schleife erfüllt ist. Ist das der Fall, wird der Rumpf der Schleife mit dem aktuellen Wert von `$i` (Ausgabe: `Zeile 2`) durchlaufen usw. Nach dem fünften Durchlauf wird `$i` auf `6` erhöht. Damit trifft die Bedingung zur Wiederholung nicht mehr zu; das Programm beendet die Schleife und läuft hinter der Schleife weiter.

Hinweis

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

[«]

1.6.2 Beispiele für Schleifen mit »for«

Einige Beispiele für Schleifensteuerungen sind in Tabelle 1.4 aufgeführt. Abläufe mit regelmäßigen Zahlenfolgen lassen sich meist mit `for`-Schleifen umsetzen. Beachten Sie bei Schleifen, die von einer größeren Zahl zu einer kleineren Zahl laufen, dass die Bedingung zur Wiederholung mithilfe des Operators `>` (oder `>=`) gebildet werden muss.

Kopf der for-Schleife	Zur Verfügung stehende Werte
`for($i=10; $i<=15; $i++)`	`10, 11, 12, 13, 14, 15`
`for($i=10; $i<15; $i++)`	`10, 11, 12, 13, 14`

Tabelle 1.4 Beispiele für Schleifensteuerungen

Kopf der for-Schleife	Zur Verfügung stehende Werte
`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<12.9; $i=$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`

Tabelle 1.4 Beispiele für Schleifensteuerungen (Forts.)

Fließkommazahlen werden nicht mathematisch exakt gespeichert. Aus diesem Grund wird im vorletzten Beispiel für die Bedingung zur Wiederholung der Wert `12.9` statt `13.0` gewählt. Das ist ein Wert, der auf »halbem Weg« zwischen dem letzten erlaubten Wert und dem ersten nicht mehr erlaubten Wert steht. Die fortlaufende Addition von `0.2` könnte statt des Werts `13.0` den Wert `12.99999` ergeben. In diesem Fall würde dieser Wert ebenfalls noch zur Verfügung stehen.

[ ] **Übung »u_for«**

Schreiben Sie ein Programm (Datei *u_for.php*), in dem mithilfe verschiedener `for`-Schleifen die in Abbildung 1.46 angegebenen Zeilen ausgegeben werden. Ein Tipp: Versuchen Sie nicht, die gesamte Übung auf einmal zu entwickeln. Schreiben Sie zunächst die erste `for`-Schleife, die die Ausgabe in der ersten Zeile erzeugt. Testen Sie anschließend das Programm. Fahren Sie erst mit der Entwicklung des nächsten Programmteils fort, falls der vorherige Programmteil fehlerfrei läuft. Für die letzte Zahlenreihe wird eine zusätzliche Verzweigung mit `if` benötigt.

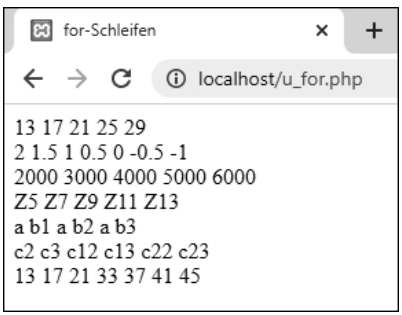


Abbildung 1.46 Ergebnis der Übung »u_for«

[«]

Hinweis

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

1.6.3 Verschachtelte Schleife mit »for«

Schleifen können verschachtelt werden. Dabei befindet sich eine Schleife innerhalb einer anderen Schleife (Schachtelung). Dadurch wird später die Bearbeitung einer zweidimensionalen Struktur möglich, zum Beispiel einer Tabelle oder eines zweidimensionalen Feldes (siehe Abschnitt 8.10). Ein Beispiel:

```
<!DOCTYPE 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 "<br>";
   }
?>
</body></html>
```

Listing 1.34 Die Datei »for_schachtel.php«

Die erste (äußere) Schleife wird fünfmal durchlaufen. Innerhalb dieser Schleife befindet sich wiederum eine (innere) Schleife, die bei jedem Durchlauf der äußeren Schleife dreimal durchlaufen wird. Anschließend wird ein Umbruch erzeugt. Es ergeben sich insgesamt 5 × 3 = 15 Wiederholungen. Abbildung 1.47 zeigt die Programmausgabe.

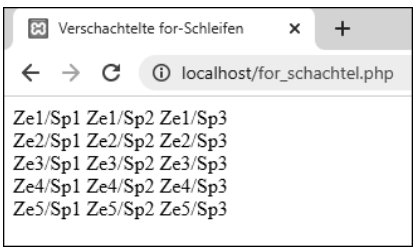


Abbildung 1.47 Verschachtelte Schleife

Übung »u_for_schachtel«

Schreiben Sie ein Programm (Datei *u_for_schachtel.php*), in dem mithilfe zweier verschachtelter `for`-Schleifen das »kleine Einmaleins« ausgegeben wird. Die Ausgabe soll so aussehen wie in Abbildung 1.48.

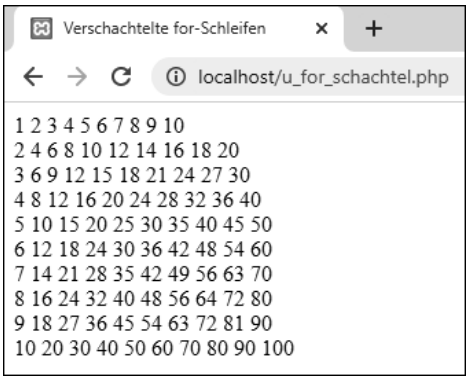


Abbildung 1.48 »Kleines Einmaleins«

1.6.4 Schleifen und Tabellen

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

```
<!DOCTYPE html>...<head>
<style>table,td {border:1px solid black;}</style></head><body>
<table>
<?php
   for($i=8; $i<=13; $i++)
   {
      echo "<tr>";
      echo "<td>Zeile</td>";
      echo "<td style='text-align:right;'>$i</td>";
      echo "</tr>";
   }
?>
</table>
</body></html>
```

Listing 1.35 Die Datei »tabelle.php«

Im Dokumentkopf wird mithilfe der CSS-Style-Angabe `border` ein Rahmen mit der Dicke `1px`, mit dem Typ »durchgezogene Linie« und der Farbe »Schwarz« für die Tabelle und ihre Zellen gesetzt.

Der Anfang und das Ende der Tabelle 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.

Die Ausgabe sehen Sie in Abbildung 1.49.

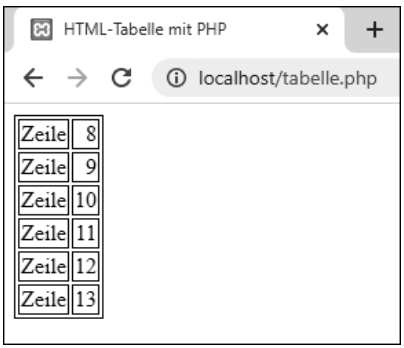


Abbildung 1.49 Schleife und Tabelle

Hinweis

Die CSS-Style-Angabe muss innerhalb der Zeichenkette (die zwischen doppelten Hochkommata steht) in einfachen Hochkommata angegeben werden, da in PHP ansonsten die Zeichenkette zu früh beendet würde.

[«]

Das erweiterte Beispiel aus der Datei *for_schachtel.php* mit einer verschachtelten Schleife in einer Tabellenstruktur sieht wie folgt aus:

```
<!DOCTYPE html>...<head>
<style>table,td {border:1px solid black;}</style></head><body>
<table>
<?php
   for($z=8; $z<=13; $z=$z+1)
   {
      echo "<tr>";
      for($s=1; $s<=5; $s=$s+1)
         echo "<td style='text-align:right;'>$z/$s</td>";
      echo "</tr>";
```

```
    }
?>
</table>
</body></html>
```

Listing 1.36 Die Datei »tabelle_schachtel.php«

Der Anfang und das Ende der Tabelle werden hier wiederum im HTML-Bereich angegeben. Die äußere Schleife sorgt für das Erzeugen der Tabellenzeilen, die innere Schleife für das Erzeugen und Füllen der Zellen. Abbildung 1.50 zeigt die Ausgabe.

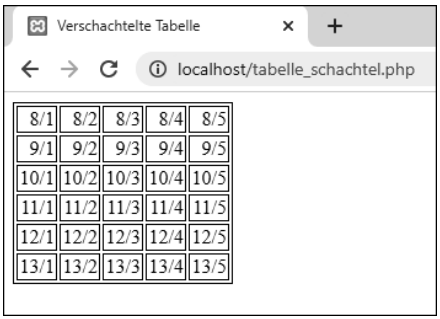


Abbildung 1.50 Verschachtelte Schleife und Tabelle

[ ] **Übung »u_tabelle«**

Erweitern Sie das Programm aus der Übung *u_for_schachtel*. Betten Sie das »kleine Einmaleins« in eine Tabelle ein (*u_tabelle.php*). Die Ausgabe soll so aussehen wie in Abbildung 1.51.

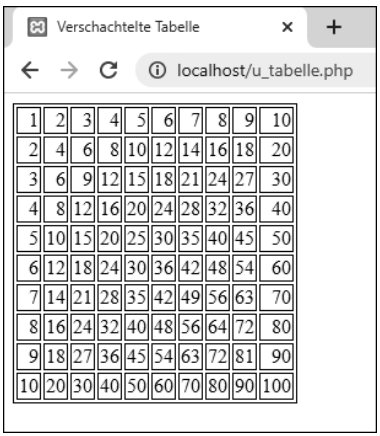


Abbildung 1.51 »Kleines Einmaleins« in einer Tabelle

1.6.5 Schleife mit »while«

Die `while`-Schleife wird dazu benutzt, eine unbestimmte Anzahl an Wiederholungen zu erzeugen. Das Ende der Wiederholungen wird bei einem der Schleifendurchläufe erreicht. `while`-Schleifen werden häufig bei Datenbankabfragen eingesetzt (siehe Abschnitt 3.3).

Im folgenden Beispiel wird gewürfelt. Die gewürfelten Zahlen werden addiert. Dies wird so lange wiederholt, bis die Summe der gewürfelten Zahlen 25 beträgt oder darüber liegt.

Zum Erzeugen der »zufälligen« Würfelergebnisse wird die Funktion `random_int()` verwendet, die mit PHP 7.0 eingeführt wurde. Sie liefert eine pseudo-zufällige ganze Zahl aus einem Zahlenbereich. Der erste Parameter kennzeichnet den Beginn des Zahlenbereichs, der zweite Parameter das Ende des Zahlenbereichs. Der Zufallszahlengenerator der Funktion `random_int()` ist auch zur Verschlüsselung geeignet.

Die Anzahl der Durchläufe ist sowohl dem Entwickler als auch der Benutzerin unbekannt. Daher kann keine `for`-Schleife verwendet werden. Das Programm sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   $summe = 0;

   while($summe < 25)
   {
      $zufallszahl = random_int(1,6);
      $summe += $zufallszahl;
      echo "Zahl $zufallszahl, Summe $summe<br>";
   }
?>
</body></html>
```

Listing 1.37 Die Datei »while.php«

Nach dem Schlüsselwort `while` folgt eine Bedingung, die wie bei einer Verzweigung in Klammern stehen muss. Beachten Sie, dass nach den Klammern kein Semikolon folgt.

Bei der ersten Prüfung der Bedingung hat `$summe` noch den Wert `0`, deshalb darf die Schleife durchlaufen werden. Innerhalb der Schleife wird der Würfelwert mithilfe des kombinierten Zuweisungsoperators (siehe Abschnitt 1.2.5) zur Variablen `$summe` addiert. Der Würfelwert und die aktuelle Zwischensumme werden ausgegeben.

Es wird wiederum zu Beginn der Schleife überprüft, ob die Summe kleiner als 25 ist. Ist das der Fall, wird die Schleife erneut durchlaufen. Andernfalls wird mit der Anweisung hinter dem Schleifenende fortgefahren. Steht dort keine Anweisung mehr, ist das Programm zu Ende. Es wird also so lange eine Zahl addiert, bis die Bedingung für die Wiederholung nicht mehr erfüllt ist. Die Seite könnte, abhängig von den zufällig ermittelten Werten, so wie in Abbildung 1.52 aussehen.

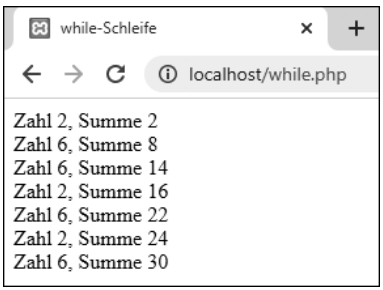


Abbildung 1.52 »while«-Schleife mit Zufallszahlen

Übung »u_while«

Erstellen Sie ein kleines Computerspiel. Zwei Spieler würfeln mithilfe des Zufallszahlengenerators gegeneinander. Die Würfe jedes Spielers sollen addiert werden. Sobald einer der beiden Spieler oder beide Spieler nach einer Spielrunde den Wert 25 erreicht oder überschritten haben, ist das Spiel zu Ende (Datei *u_while.php*).

Der Gewinner ist der Spieler mit der höheren Punktzahl. Seine Nummer soll anschließend ausgegeben werden. Die Ausgabe könnte so wie in Abbildung 1.53 aussehen.

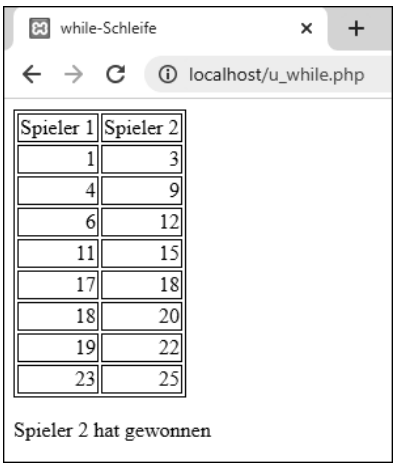


Abbildung 1.53 Übung »u_while«, Spiel

1.6.6 Schleife mit »do-while«

Die `do-while`-Schleife arbeitet wie die `while`-Schleife, es gibt aber einen wichtigen Unterschied: Die Prüfung für die Wiederholung wird erst am Ende der Schleife durchgeführt. Die Schleife wird also mindestens einmal ausgeführt. Das Würfelprogramm sieht daher wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   $summe = 0;

   do
   {
      $zufallszahl = random_int(1,6);
      $summe += $zufallszahl;
      echo "Zahl $zufallszahl, Summe $summe<br>";
   }
   while($summe < 25);
?>
</body></html>
```

Listing 1.38 Die Datei »dowhile.php«

Im Unterschied zur `while`-Schleife folgt nach der Bedingung der `do-while`-Schleife ein Semikolon. Die Schleife wird selbst für den Fall ausgeführt, dass die Variable `$summe` bereits vor der Schleife den Wert `25` oder mehr hat.

1.6.7 Abbruch einer Schleife mit »break«

Mithilfe der Anweisung `break`, die Ihnen bereits aus der `switch`-Verzweigung bekannt ist, kann eine Schleife vorzeitig beendet werden. Damit wird eine zusätzliche Möglichkeit für eine Schleifensteuerung geschaffen, mit deren Hilfe Sie ein Programm lesbarer machen können.

Hinweis [«]

Eine `break`-Anweisung in einer Schleife tritt immer gemeinsam mit einer Bedingung auf, da der vorzeitige Abbruch einer Schleife nur in einem »Sonderfall« erfolgen sollte.

Im folgenden Beispiel wird wiederum gewürfelt, solange die Summe kleiner als 25 ist. Allerdings soll höchstens sechsmal gewürfelt und gegebenenfalls abgebrochen werden.

```
<!DOCTYPE html>...<body>
<?php
   $summe = 0;
   $zaehler = 0;

   while($summe < 25)
   {
      $zufallszahl = random_int(1,6);
      $summe += $zufallszahl;
      $zaehler++;
      echo "Nr. $zaehler, Zahl $zufallszahl,";
      echo " Summe $summe<br>";
      if($zaehler >= 6)         // Sonderfall
         break;
   }
?>
</body></html>
```

Listing 1.39 Die Datei »break.php«

Es wird ein zusätzlicher Zähler (Variable `$zaehler`) verwendet. Diese Variable wird zunächst auf `0` gesetzt. Innerhalb der Schleife wird ihr Wert stets um `1` erhöht. Sie zählt also die Anzahl der Schleifendurchläufe.

Wird dabei die Zahl `6` erreicht beziehungsweise überschritten, bricht die Schleife unmittelbar ab. Dies geschieht auch, falls die Summe noch kleiner als `25` ist. Die Seite sieht, abhängig von den zufällig ermittelten Werten, so aus wie in Abbildung 1.54.

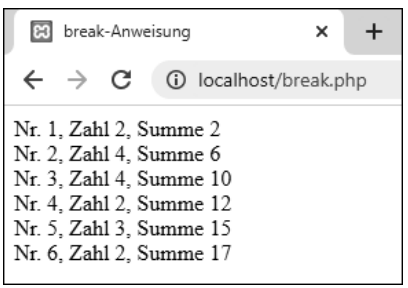


Abbildung 1.54 Beispiel zu »break«

1.6.8 Fortsetzung einer Schleife mit »continue«

Die Anweisung `continue` sorgt für den Abbruch des aktuellen Schleifendurchlaufs. Die Schleife wird anschließend unmittelbar mit dem nächsten Durchlauf fortgesetzt. Ein mögliches Programm sieht wie folgt aus:

```
<!DOCTYPE html>...<body>
<?php
   for($i=1; $i<=15; $i++)
   {
      if($i>=5 && $i<=12)
         continue;
      echo "Zeile $i<br>";
   }
?>
</body></html>
```

Listing 1.40 Die Datei »continue.php«

Für die Werte 5 bis 12 wird keine Ausgabe vorgenommen; dies zeigt Abbildung 1.55.

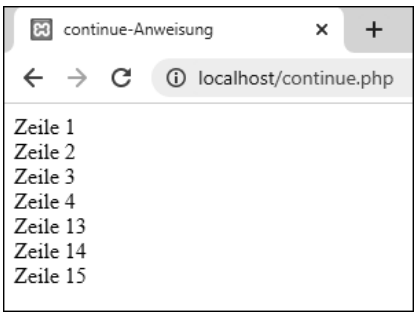


Abbildung 1.55 Beispiel zu »continue«

1.7 Funktionen

Es gibt in PHP zahlreiche vordefinierte Funktionen, die Sie einsetzen können. Darüber hinaus haben Sie die Möglichkeit, eigene Funktionen zu schreiben, sogenannte *benutzerdefinierte Funktionen*. Diese haben die folgenden Vorteile:

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

- Programme können modularisiert werden. Dies bedeutet, dass sie in kleinere Bestandteile zerlegt werden können, die übersichtlicher sind und einfacher gewartet werden können.

1.7.1 Ein erstes Beispiel

Ein Beispiel für eine einfache benutzerdefinierte Funktion:

```
<!DOCTYPE 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 1.41 Die Datei »funktion_einfach.php«

Eigene Funktionen werden mithilfe von `function ... () { ... }` definiert. Der Name der Funktion folgt nach dem Schlüsselwort `function`, und in runden Klammern folgen die Parameter, sofern welche vorhanden sind. Anschließend folgt in geschweiften Klammern der eigentliche Funktionsrumpf. Häufig wird die Funktion im Kopf eines HTML-Dokuments definiert, wie hier bei der Funktion `trennstrich()`.

Die Aufgabe der Funktion `trennstrich()` ist die Darstellung eines Zeilenumbruchs, von `40` Bindestrichen und eines weiteren Zeilenumbruchs. Jedes Mal, wenn sie vom eigentlichen Programm im Rumpf des Dokuments (mit `trennstrich()`) aufgerufen wird, führt sie die genannte Aufgabe aus. Die Ausgabe sehen Sie in Abbildung 1.56.

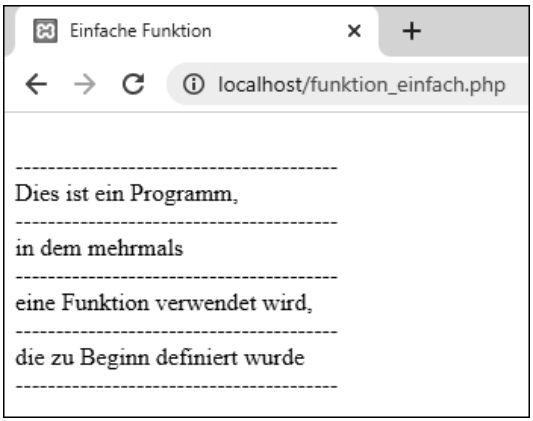


Abbildung 1.56 Mehrere Aufrufe einer Funktion

Übung »u_funktion_einfach«

Erstellen Sie eine Funktion `vermerk()`, die einen Entwicklervermerk erzeugt: Jedes Mal, wenn die Funktion aufgerufen wird, erscheint der Name des Entwicklers in einer Tabellenzelle mit Rahmen. Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals aufgerufen wird (Datei *u_funktion_einfach.php*). Das Ergebnis könnte so aussehen wie in Abbildung 1.57.

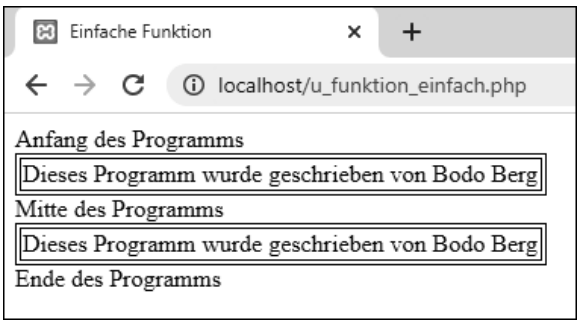


Abbildung 1.57 Ergebnis der Übung »u_funktion_einfach«

1.7.2 Definition, Aufruf und Funktionstypen

Der Aufruf einer eigenen oder einer vordefinierten Funktion erfolgt

- entweder aus dem Rumpf des Dokuments heraus (im zuvor angegebenen Beispiel mit `trennstrich()`) oder
- aus anderen Funktionen heraus.

Dabei ist der Ort der Funktionsdefinition wichtig. Sie können nur Funktionen aufrufen, die dem Programm bekannt sind. Diese müssen also

- entweder zu den vordefinierten Funktionen gehören oder
- als eigene Funktion in der Datei definiert werden (wie im oben angegebenen Beispiel) oder
- als eigene Funktion aus einer externen Datei stammen, die in dieser Datei verfügbar ist (siehe Abschnitt 1.10.7).

Für Funktionen mit Parametern bzw. Rückgabewert gilt Folgendes:

- Eine Funktion ohne Parameter führt bei jedem Aufruf immer genau die gleiche Aufgabe aus (wie im oben angegebenen Beispiel).
- Eine Funktion mit einem oder mehreren Parametern führt bei jedem Aufruf in Abhängigkeit von den Parametern ähnliche Aufgaben aus.
- Eine Funktion mit einem Rückgabewert führt gleiche oder ähnliche Aufgaben aus und liefert ein Ergebnis an die aufrufende Stelle zurück.

Für den Namen einer Funktion gelten die gleichen Regeln wie für den Namen einer Variablen (siehe Abschnitt 1.2.2). Der einzige Unterschied besteht darin, dass Namen von Funktionen nicht mit dem Dollarzeichen `$` beginnen.

Der Name einer Funktion muss innerhalb eines Programms eindeutig sein. Sie können also eine bereits vordefinierte oder eine eigene Funktion nicht noch einmal definieren.

1.7.3 Funktionen mit einem Parameter

Eine Funktion mit einem Parameter führt bei jedem Aufruf in Abhängigkeit vom Parameterwert ähnliche Aufgaben aus. Das vorherige Beispiel wird nun ein wenig erweitert. Die Funktion erzeugt unterschiedlich lange Trennstriche, wie Sie nachfolgend erkennen können:

```
<!DOCTYPE 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 1.42 Die Datei »funktion_parameter.php«

Die Funktion `trennstrich()` wird insgesamt viermal aufgerufen, jedes Mal mit einem anderen Wert innerhalb der Klammern hinter dem Namen der Funktion. Dies ist der Parameter. Es kann sich dabei um eine Zahl, eine Variable oder das Ergebnis einer Berechnung handeln.

Der Parameter 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 beinahe die gleiche Aktion durchgeführt, jeweils in Abhängigkeit vom Wert des Parameters. Abbildung 1.58 zeigt die Ausgabe.

Bei dem Parameter, der an die Variable `$anzahl` übergeben wird, sollte es sich um eine ganze Zahl handeln. Fließkommazahlen, Zeichenketten oder andere Typen von Variablen sind hier nicht sinnvoll. Seit PHP 7.0 gibt es die Möglichkeit, die Datentypen von Parametern beim Aufruf einer Funktion mithilfe von Typhinweisen strenger zu kontrollieren (siehe Abschnitt 1.10.11). Dies trägt zur Verbesserung Ihrer Programme bei.

Übergeben Sie einer Funktion beim Aufruf zu wenige Parameter, erhalten Sie seit PHP 7.1 die Fehlermeldung `Uncaught ArgumentCountError: Too few arguments` .... Übergeben Sie zu viele Parameter, werden die überzähligen Parameter ignoriert.

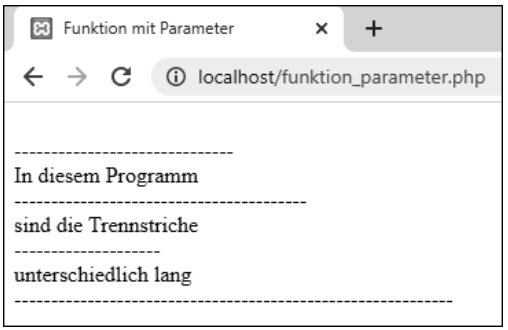


Abbildung 1.58 Aufrufe mit unterschiedlichen Parametern

Übung »u_funktion_parameter1«

Erweitern Sie die Funktion `vermerk()` aus der Übung *u_funktion_einfach*. 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 es in Abbildung 1.59 dargestellt ist (Datei *u_funktion_parameter1.php*).

Testen Sie Ihre Funktion mit einem geeigneten Programm, in dem die Funktion mehrmals mit verschiedenen Namen aufgerufen wird.

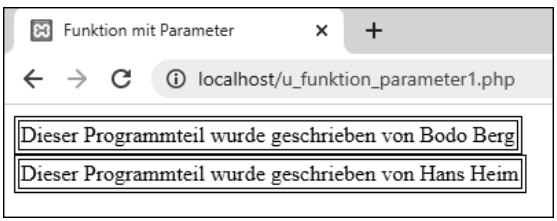


Abbildung 1.59 Ergebnis der Übung »u_funktion_parameter1«

Übung »u_funktion_parameter2«

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 *u_funktion_parameter2.php*). In Abbildung 1.60 sehen Sie ein Beispiel.

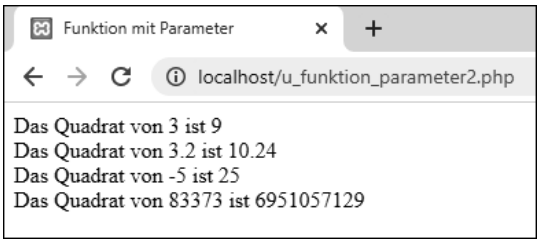


Abbildung 1.60 Ergebnis der Übung »u_funktion_parameter2«

1.7.4 Funktionen mit mehreren Parametern

Werden einer Funktion mehrere Parameter übergeben, sind die Anzahl, der Datentyp (Zahl oder Zeichenkette) und die Reihenfolge der Parameter wichtig. Der erste Wert wird an den ersten Parameter übergeben, der zweite Wert an den zweiten Parameter usw. Hier sehen Sie ein Beispiel für eine eigene Funktion mit mehreren Parametern:

```
<!DOCTYPE html>...<head>...
<?php
   function flexloop($von, $bis, $schritt)
   {
      echo "Eine Schleife von $von bis $bis mit der Schrittweite $schritt<br>";
      for($i = $von; $i <= $bis; $i = $i + $schritt)
         echo "$i ";
   }
?>
</head><body>
<?php
   echo "<p>Nummer 1:<br>";
   flexloop(5,27,3);

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

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

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

Listing 1.43 Die Datei »funktion_mehrere.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.

Die Variablen werden zur Steuerung der `for`-Schleife in der Funktion verwendet. Es wird also bei jedem Aufruf eine ähnliche Aktion durchgeführt, beeinflusst von den Werten der Parameter. Die Ausgabe sieht so aus wie in Abbildung 1.61.

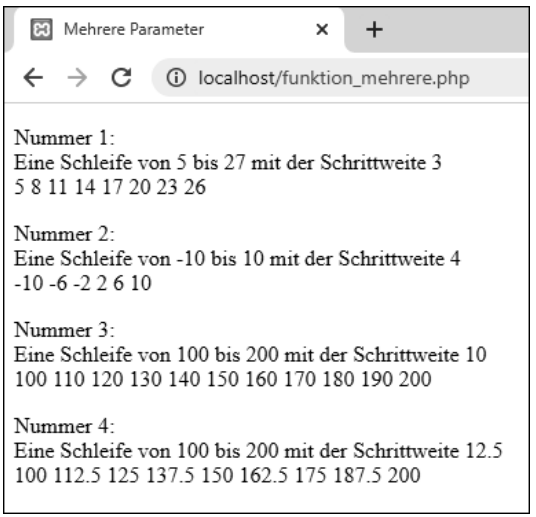


Abbildung 1.61 Funktion mit mehreren Parametern

Übung »u_funktion_mehrere1«

Schreiben Sie ein Programm (Datei *u_funktion_mehrere1.php*), in dem eine Funktion `mittel()` definiert und benutzt wird, die den arithmetischen Mittelwert von drei Zahlen berechnet und ausgibt. Zur Berechnung bilden Sie die Summe der drei Zahlen und teilen diese anschließend durch 3.

Die drei Zahlen werden der Funktion jeweils als Parameter übergeben. Testen Sie die Funktion mit mehreren verschiedenen Aufrufen innerhalb des Programms. Die Ausgabe könnte so aussehen wie Abbildung 1.62.

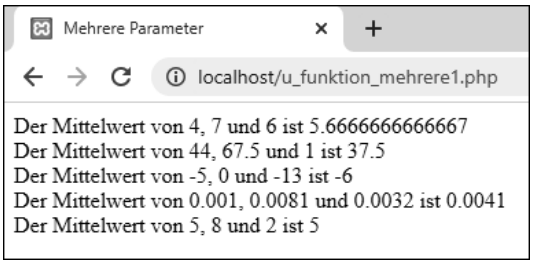


Abbildung 1.62 Ergebnis der Übung »u_funktion_mehrere1«

Übung »u_funktion_mehrere2« [✎]

Erweitern Sie die Funktion `vermerk()` aus der Übung *u_funktion_parameter1*. Sie soll von verschiedenen Entwicklern und Entwicklerinnen 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 gemäß der folgenden Regel 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 *u_funktion_mehrere2.php*). Eine mögliche Ausgabe sehen Sie in Abbildung 1.63.

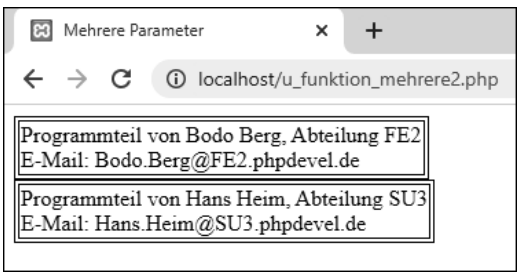


Abbildung 1.63 Ergebnis der Übung »u_funktion_mehrere2«

1.7.5 Rückgabewert einer Funktion

Viele eigene und vordefinierte Funktionen arbeiten auf die folgende Weise: Innerhalb der Funktion wird ein Ergebnis ermittelt und mithilfe eines sogenannten Rückgabewerts an die aufrufende Stelle zurückgeliefert. Dieser Wert muss entweder in einer Vari-

ablen gespeichert oder direkt ausgegeben werden, andernfalls geht er verloren. Hier sehen Sie ein Beispiel für eine Funktion mit einem Rückgabewert:

```
<!DOCTYPE html>...<head>...
<?php
   function add($z1, $z2)
   {
      $summe = $z1 + $z2;
      return $summe;
   }
?>
</head><body>
<?php
   $c = add(3,4);     /* Aufruf und Zuweisung */
   echo "Summe: $c<br>";

   $x = 5;
   $c = add($x,12);   /* Aufruf und Zuweisung */
   echo "Summe: $c<br>";

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

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

Listing 1.44 Die Datei »funktion_rueckgabewert.php«

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

Mithilfe der Anweisung `return` wird der Wert an die aufrufende Stelle zurückgeliefert und kann dort weiterverarbeitet werden. In den ersten beiden Fällen wird der Wert in der Variablen `$c` gespeichert, und im dritten Fall wird er ohne Zwischenspeicherung direkt ausgegeben. Die Ausgabe sehen Sie in Abbildung 1.64.

Der Aufruf einer Funktion darf nicht innerhalb einer Zeichenkette stehen. Die letzte Zeile der Ausgabe zeigt, dass in diesem Fall nur der Name der Funktion und ihre Parameter genannt werden, die Funktion selbst aber nicht aufgerufen wird.

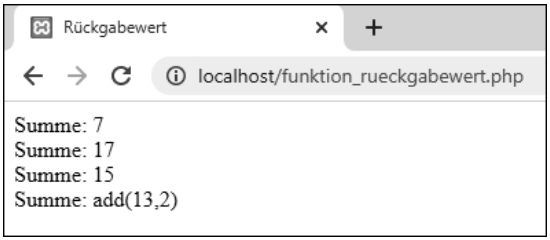


Abbildung 1.64 Funktion mit Rückgabewert

Mithilfe der Anweisung `return` kann eine Funktion auch vorzeitig verlassen werden und nicht erst am Ende. Dies gilt unabhängig davon, ob sie einen Wert zurückliefert oder nicht.

Bei dem Rückgabewert, den die Funktion `add()` liefert, sollte es sich um eine Zahl handeln. Eine Zeichenkette oder ein anderer Typ von Variablen ist hier nicht sinnvoll. Seit PHP 7.0 gibt es die Möglichkeit, die Datentypen von Rückgabewerten mithilfe von Typhinweisen strenger zu kontrollieren (siehe Abschnitt 1.10.11). Dies trägt zur Verbesserung Ihrer Programme bei.

Übung »u_funktion_rueckgabewert« [ ]

Schreiben Sie ein Programm (Datei *u_funktion_rueckgabewert.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 Fall:

```
Maximum: 4
```

1.7.6 Kopie und Referenz

Bei der Übergabe von Parametern an eine Funktion müssen Sie sich noch die folgende Frage stellen: Was passiert, wenn ich in der Funktion einen der soeben übergebenen Parameter verändere?

PHP bietet hier zwei Möglichkeiten an:

- **Übergabe der Parameter an eine Kopie (*Call-by-Value*):** Eine Veränderung der Kopie hat *keine* Rückwirkung auf das Original. Diese Methode wird zum Beispiel angewendet, wenn die Daten nur in eine Richtung fließen, also nur Werte an die Funktion übergeben werden. So wurde es bei den bisherigen Programmen für Funktionen in diesem Buch gemacht.
- **Übergabe der Parameter an eine Referenz (*Call-by-Reference*):** Eine Veränderung hat eine Rückwirkung auf das Original. Diese Methode wird angewendet, wenn die Funktion mehr als einen Wert ermitteln und liefern soll. Über einen Rückgabewert (siehe Abschnitt 1.7.5) könnte nur ein einziger Wert zurückgeliefert werden.

Beide Methoden sollen zum Vergleich am selben Beispiel dargestellt werden. Den beiden Funktionen `rtauschen()` und `vtauschen()` werden jeweils zwei Parameter übergeben. Innerhalb der Funktionen sollen die beiden übergebenen Parameter miteinander vertauscht werden.

In Abhängigkeit von den verschiedenen angewendeten Methoden wird dieser Tauschvorgang Rückwirkungen auf die Originalvariablen im Hauptprogramm haben. Die Werte werden jeweils vor und nach dem Tauschvorgang angezeigt:

```
<!DOCTYPE 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 "<p>Per Kopie, vorher: $x, $y<br>";
    vtauschen($x,$y);
    echo "Per Kopie, nachher: $x, $y</p>";

    $x = 12;   $y = 18;
    echo "<p>Per Referenz, vorher: $x, $y<br>";
    rtauschen($x,$y);
    echo "Per Referenz, nachher: $x, $y</p>";
?>
</body></html>
```

Listing 1.45 Die Datei »funktion_referenz.php«

Die Anwendung der beiden Methoden ergibt Folgendes:

- **Per Kopie:** Der Wert der Variablen `$x` wird beim Aufruf der Funktion `vtauschen()` an die Variable `$a` übergeben. Der Wert der Variablen `$y` wird an die Variable `$b` übergeben. `$a` und `$b` sind Kopien von `$x` und `$y`. Innerhalb der Funktion `vtauschen()` werden die Werte von `$a` und `$b` getauscht. Da eben nur die Werte der Kopien getauscht werden, hat dies keine Auswirkungen auf die Werte der Originale `$x` und `$y`.
- **Per Referenz:** Den Unterschied sehen Sie im Funktionskopf `function rtauschen(&$a, &$b)`. Die Variable `$x` wird beim Aufruf der Funktion `rtauschen()` an die Referenz `$a` übergeben (siehe auch Abschnitt 1.2.9). Die Variable `$y` wird an die Referenz `$b` übergeben. Über die beiden Referenzen kann jederzeit direkt auf die beiden Originale `$x` und `$y` zugegriffen werden. Innerhalb der Funktion werden die Werte der Referenzen vertauscht. Dadurch werden auch die Werte der Originale `$x` und `$y` vertauscht.

Die Ausgabe, jeweils mit den Werten vor und nach der Vertauschung, sieht so aus wie in Abbildung 1.65.

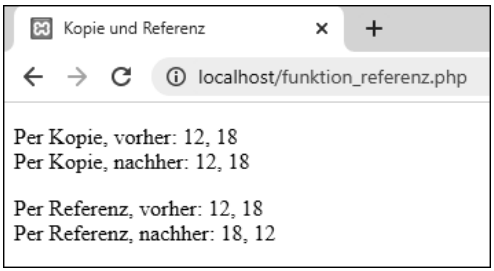


Abbildung 1.65 Kopie und Referenz

[ ] **Übung »u_funktion_referenz«**

Schreiben Sie ein PHP-Programm (Datei *u_funktion_referenz.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 und 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 zweite Methode (Call-by-Reference). Nach einem Funktionsaufruf mit den Parametern `5` und `7` und der anschließenden Ausgabe erscheint eine Ausgabe wie in Abbildung 1.66.

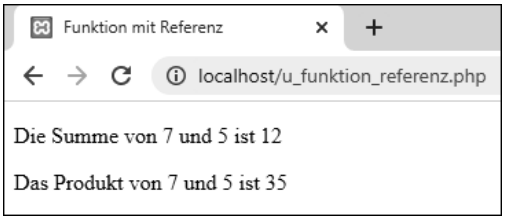


Abbildung 1.66 Ergebnis der Übung »u_funktion_referenz«

1.7.7 Gültigkeitsbereich von Variablen

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

- **Lokale Variablen:** Diese werden innerhalb einer Funktion definiert und stehen nur innerhalb dieser Funktion zur Verfügung.
- **Globale Variablen:** Diese werden außerhalb einer Funktion definiert. Sie gelten nur außerhalb von Funktionen, außer wenn sie innerhalb einer Funktion nochmals mit dem Schlüsselwort `global` deklariert werden. Dies ist ein Unterschied zu vielen anderen Programmiersprachen.
- **Superglobale Variablen:** Bei diesen Variablen handelt es sich um PHP-Systemvariablen. Sie stehen sowohl innerhalb als auch außerhalb von Funktionen zur Verfügung. Zu ihnen zählt das assoziative Feld `$_POST`, das die Namen und Werte von Formularfeldern zur Verfügung stellt.

Im Zusammenhang mit dem Gültigkeitsbereich von Variablen gelten folgende Regeln:

- Ein Parameter, der als Kopie an eine Funktion übergeben wird, ist dort lokal.
- Lokale Variablen gleichen Namens in unterschiedlichen Funktionen haben nichts miteinander und auch nichts mit einer globalen Variablen gleichen Namens zu tun.

- Möchten Sie eine globale Variable innerhalb einer Funktion benutzen, muss sie dort entweder mit dem Schlüsselwort `global` bekannt gemacht oder als Parameter übergeben werden.

Hinweis [«]

Die Variablen eines Programms sollten immer »so lokal wie möglich« und »so wenig global wie möglich« sein. Das bietet folgende Vorteile:

- Die Modularisierung des Programms wird verbessert, das heißt die Zerlegung eines Programms in übersichtliche Programmteile mit klar definierten Schnittstellen zwischen den Teilen.
- Die Wiederverwendbarkeit der Funktionen für andere Programme wird erleichtert.
- Die Variablen können nicht so leicht aus Versehen an weit voneinander entfernten Stellen verändert werden.

Hier sehen Sie ein Beispiel mit lokalen und globalen Variablen sowie dem Schlüsselwort `global`:

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

Listing 1.46 Die Datei »funktion_global.php«

In diesem Programm existieren fünf unterschiedliche Variablen.

Betrachten wir zunächst die beiden lokalen Variablen `$y` und `$z`, die nur innerhalb der Funktion `summiere()` bekannt sind: Zum Zeitpunkt des ersten Ausgabebefehls in der Funktion existiert `$z` noch nicht. Daher kann für `$z` kein Wert ausgegeben werden. Anschließend erhalten `$y` und `$z` innerhalb der Funktion einen Wert. `$z` kann nun ausgegeben werden. Nach dem Verlassen der Funktion `summiere()` sind beide Werte nicht mehr verfügbar.

Im Hauptprogramm gibt es insgesamt drei globale 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 beispielsweise innerhalb der Funktion `summiere()`.

Die Ausgabe des Programms sehen Sie in Abbildung 1.67.

[»]

Hinweis

Der Umfang der Anmerkungen, Warnungen und Fehlermeldungen, die vom System angezeigt werden, hängt von den Einstellungen in der Konfigurationsdatei *php.ini* ab. Mehr dazu folgt in Abschnitt 5.1. Bei mir ist XAMPP im Verzeichnis *D:\xampp* installiert.

Abbildung 1.67 Lokale und globale Variablen

1.8 Behandlung von Fehlern

Bei der Behandlung von Fehlern kommt das Konzept des *Exception Handlings* (deutsch: *Ausnahmebehandlung*) zum Einsatz. Es bietet die Möglichkeit, bestimmte Fehler abzufangen. Damit wird es einem Programm ermöglicht, weiterzulaufen, obwohl ein Fehler aufgetreten ist. Sie werden lediglich über den Fehler informiert, sodass Sie die Fehlerursache abstellen können.

Beim Exception Handling wird der Codebereich, in dem ein Fehler auftreten kann, in einen sogenannten `try`-Block eingeschlossen. Es wird »versucht«, den Code auszuführen. Tritt ein definierter Fehler auf, wird ein Objekt der Klasse `Exception` durch die Anweisung `throw` erzeugt. (Zum Thema »Klassen und Objekte« finden Sie in Kapitel 4 eine ausführliche Beschreibung.)

Anschließend wird statt des restlichen Codes im `try`-Block der Code im zugehörigen `catch`-Block ausgeführt; der Fehler wird somit »abgefangen«. Dies erläutere ich nachfolgend am Beispiel eines Programms, das drei Fehler enthält.

1.8.1 Ohne Ausnahmebehandlung

Zunächst betrachten wir die Version des Programms ohne Ausnahmebehandlung. Es werden Warnungen und Fehlermeldungen angezeigt. Zudem kommt es zu einem Programmabbruch:

```
<!DOCTYPE html>...<body>
<?php
   /* Variable unbekannt */
   echo "Variable: $x<br>";

   /* Division durch 0 */
   $x = 42;
   $y = 0;
   $z = $x / $y;
   echo "Division: $x / $y = $z<br>";

   /* Zugriff auf Funktion */
   testFunktion();

   echo "Ende des Programms";
?>
</body></html>
```

Listing 1.47 Die Datei »exception_ohne.php«

Als Erstes wird eine Variable verwendet, die bis zu diesem Zeitpunkt noch unbekannt ist. Anschließend wird eine Division durch 0 durchgeführt. Das ist mathematisch nicht erlaubt und führt in PHP zur Ausgabe von `INF`. Das steht abkürzend für *infinity* (deutsch: *unendlich*). Als Letztes wird eine unbekannte Funktion aufgerufen.

Ein schwerwiegender Fehler (englisch: *fatal error*) führt zu einem vorzeitigen Abbruch des Programms, siehe auch Abbildung 1.68. Die restlichen Teile des Programms werden in diesem Fall nicht mehr ausgeführt.

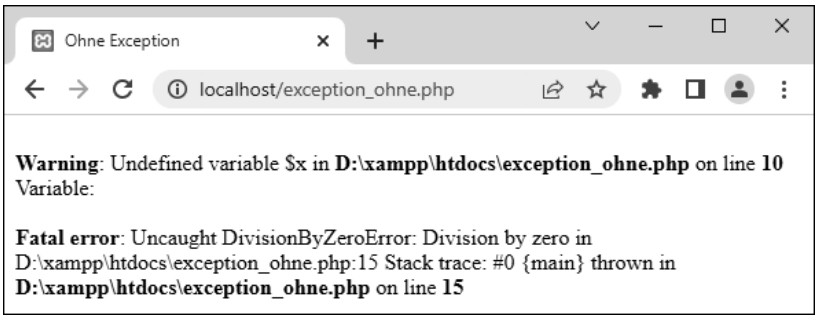


Abbildung 1.68 Ausgabe ohne Ausnahmebehandlung

[»]

Hinweis

Auch hier gilt wieder: Der Umfang der Anmerkungen, Warnungen und Fehlermeldungen, die vom System angezeigt werden, hängt von den Einstellungen in der Konfigurationsdatei *php.ini* ab.

1.8.2 Mit Ausnahmebehandlung

Nun folgt die Version des Programms mit Ausnahmebehandlung:

```
<!DOCTYPE html>...<body>
<?php
   /* Variable unbekannt */
   try
   {
      if(!isset($x))
         throw new Exception("Variable unbekannt");
      echo "Variable: $x<br>";
   }
   catch(Exception $e)
   {
      echo $e->getMessage() . "<br>";
   }
   finally
   {
```

```
      echo "Ende, Variable unbekannt<br>";
   }

   /* Division durch 0 */
   $x = 42;
   $y = 0;
   try
   {
      if($y == 0)
         throw new Exception("Division durch 0");
      $z = $x / $y;
      echo "Division: $x / $y = $z<br>";
   }
   catch(Exception $e)
   {
      echo $e->getMessage() . "<br>";
   }

   /* Zugriff auf Funktion */
   try
   {
      if(!function_exists("testFunktion"))
         throw new Exception("Funktion unbekannt");
      testFunktion();
   }
   catch(Exception $e)
   {
      echo $e->getMessage() . "<br>";
   }

   echo "Ende des Programms";
?>
</body></html>
```

Listing 1.48 Die Datei »exception_mit.php«

Es beginnt mit der Nutzung der unbekannten Variablen: Vor ihrer Nutzung wird mithilfe der Funktion `isset()` geprüft, ob die Variable existiert. Das Ganze findet in einem `try`-Block statt. Es ist also ein »Versuch«, die Variable zu nutzen.

Wird festgestellt, dass die Variable nicht existiert, wird zunächst mithilfe des Schlüsselworts `new` ein Objekt der Klasse `Exception` mit einem passenden Text erzeugt. Dieses `Exception`-Objekt (kurz: diese Exception) wird mithilfe des Schlüsselworts `throw` »geworfen«. Mehr zu Objekten und Klassen finden Sie in Kapitel 4.

Die Exception wird in demjenigen `catch`-Block »gefangen«, der dem `try`-Block zugeordnet ist. Bei der Erzeugung der Exception wird ihr eine Meldung mitgegeben. Diese Fehlermeldung wird mithilfe der Methode `getMessage()` ausgegeben (siehe Abbildung 1.69). Der Rest des `try`-Blocks wird in diesem Fall nicht mehr bearbeitet.

Sie können optional einen `finally`-Block anfügen. Die darin enthaltenen Anweisungen werden in jedem Fall durchgeführt, unabhängig davon, ob eine Ausnahme aufgetreten ist oder nicht.

Wird das `Exception`-Objekt selbst nicht benötigt, kann der Name der Variablen im `catch`-Block seit PHP 8.0 weggelassen werden. Der `catch`-Block würde dann nur noch wie folgt eingeleitet werden: `catch(Exception) { ... }`

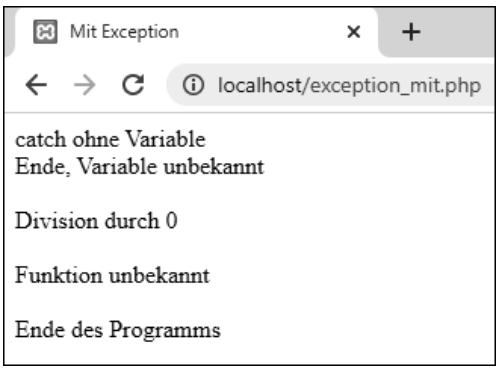


Abbildung 1.69 Ausgabe mit Ausnahmebehandlung

Es folgt die Division durch 0: Es wird geprüft, ob durch `0` geteilt werden soll. Ist dies der Fall, wird eine entsprechende Meldung ausgegeben.

Als Letztes geht es um die unbekannte Funktion: Mithilfe der Funktion `function_exists()` wird geprüft, ob die Funktion existiert. Ist das nicht der Fall, wird eine entsprechende Meldung ausgegeben.

Es erfolgt kein Programmabbruch. Das Programm läuft bis zum Ende.

1.9 Felder

Um eine größere Menge von zusammengehörigen Daten zu speichern, können Sie ein Feld von Variablen mit einem einheitlichen Namen nutzen. Felder bieten eine schnelle und komfortable Verarbeitung. Im Sprachgebrauch von Entwicklern und Entwicklerinnen wird ein Feld auch häufig *Array* genannt. PHP unterstützt zwei Typen von Feldern:

- **Numerisch indizierte Felder:** Die einzelnen Variablen in einem numerisch indizierten Feld werden über eine laufende Nummer innerhalb des Felds angesprochen.
- **Assoziative Felder** (auch *Hash-Tabellen* genannt)**:** Die einzelnen Variablen in einem assoziativen Feld werden über eine eindeutige Bezeichnung innerhalb des Felds angesprochen.

Die genannten Feldtypen werden in diesem Abschnitt angesprochen. Mehr zu Feldern finden Sie in Kapitel 8.

Ein eindimensionales Feld können Sie sich als Liste von zusammengehörigen Zahlen vorstellen. Das kann zum Beispiel eine Reihe von Temperaturwerten sein. Es kann sich aber auch um eine Reihe von zusammengehörigen Zeichenketten handeln, zum Beispiel um die Namen der Mitglieder einer Gruppe.

1.9.1 Numerisch indizierte Felder

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

```
<!DOCTYPE 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 "Temperatur $i: $tp[$i]<br>";
?>
</body></html>
```

Listing 1.49 Die Datei »feld_numerisch.php«

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

- Mithilfe der Funktion `array()` wird die Variable `$tp` zu einem Feld mit fünf Elementen. Diese Elemente werden automatisch durchnummeriert, beginnend bei `0`.
- Felder können auch einfach durch die Zuweisung einzelner Elemente erzeugt oder vergrößert werden. Das ist hier mit den beiden Zuweisungen `$tp[5] = 20.2;` und `$tp[6] = 16.6;` geschehen. Dabei ist die bisherige Nummerierung zu beachten, andernfalls könnten vorhandene Elemente überschrieben werden.

Ein einzelnes Feldelement sprechen Sie an, indem Sie nach dem Namen des Felds in eckigen Klammern die laufende Nummer des Elements angeben. Diese laufende Nummer wird auch *Index* genannt.

Elemente von eindimensionalen Feldern können wie einzelne Variablen direkt in Zeichenketten eingebettet werden. Gespeichert beziehungsweise ausgegeben wird der Wert des Elements.

Insgesamt hat das Feld nun sieben Elemente. Die Struktur erkennen Sie in Tabelle 1.5.

Name des Elements	Nummer (= Index) des Elements	Wert des Elements
`$tp[0]`	`0`	`17.5`
`$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`

Tabelle 1.5 Numerisch indiziertes Feld

Diese Elemente werden anschließend mithilfe einer `for`-Schleife untereinander ausgegeben. Dabei nimmt die Schleifenvariable `$i` nacheinander die verwendeten Indexwerte an (`0` bis `6`). Abbildung 1.70 zeigt die Ausgabe.

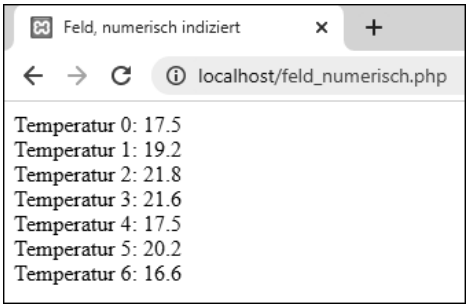


Abbildung 1.70 Numerisch indiziertes Feld

Übung »u_feld_numerisch«

Nun sollen der Vorname und das Alter von sechs Personen in zwei Feldern gespeichert werden. Das erste Feld soll die Vornamen enthalten und das zweite Feld die dazugehörigen Altersangaben. Die Elemente der beiden Felder sollen paarweise als Tabelle auf dem Bildschirm ausgegeben werden (Datei *u_feld_numerisch.php*, siehe Abbildung 1.71).

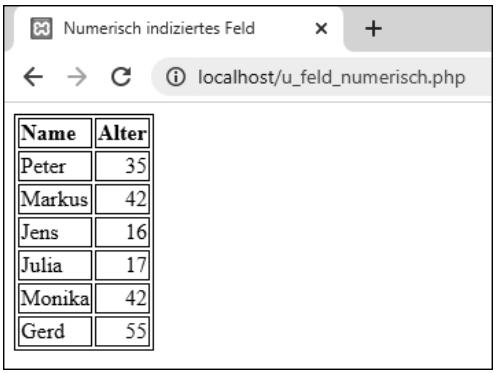


Abbildung 1.71 Ergebnis der Übung »u_feld_numerisch«

Es gibt eine alternative Schreibweise zur Erzeugung eines numerisch indizierten Feldes. Die erste Zeile des oben angegebenen Programms *numerisch.php* hätte auch so lauten können:

```
$tp = [17.5, 19.2, 21.8, 21.6, 17.5];
```

Die Werte werden innerhalb von eckigen Klammern [ ] aufgelistet und der Variablen `$tp` zugewiesen. Auch damit wird `$tp`, mit derselben Indizierung, zu einem Feld.

1.9.2 Assoziative Felder

Die Temperaturwerte aus dem vorherigen Abschnitt sollen nun in einem assoziativen Feld angeordnet werden. Die Elemente eines solchen Felds werden nicht über eine laufende Nummer, sondern über einen Schlüssel (englisch: *key*) identifiziert. Dadurch wird es möglich, den Feldelementen eindeutige Begriffe zuzuordnen. Zunächst sollen die Werte wiederum gespeichert und ausgegeben werden.

```
<!DOCTYPE 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 "<p>" . $tp["Montag"] . "</p>";

   // Alle Keys und Values aus dem Feld
   echo "<p>";
   foreach($tp as $name=>$wert)
      echo "$name, $wert<br>";
   echo "</p>";

   // Nur alle Values aus dem Feld zum Berechnen des Mittelwerts
   $summe = 0;
   foreach($tp as $wert)
      $summe = $summe + $wert;
   $mittelwert = $summe / 7;
   echo "<p>Mittelwert: $mittelwert</p>";
?>
</body></html>
```

Listing 1.50 Die Datei »feld_assoziativ.php«

Abbildung 1.72 zeigt die Ausgabe des Programms.

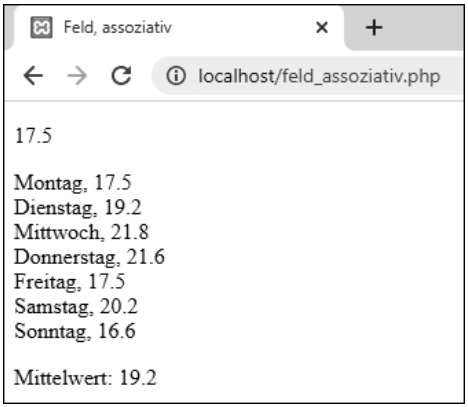


Abbildung 1.72 Assoziatives Feld

Die Verwendung assoziativer Felder erscheint zunächst etwas unübersichtlich. Wenn Sie sich aber mit der Vorgehensweise vertraut gemacht haben, können assoziative Felder einige Vorteile mit sich bringen. Es gibt zwei Techniken zur Erzeugung eines Felds:

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

Insgesamt hat das Feld nun sieben Elemente (siehe Tabelle 1.6).

Name	Schlüsselbezeichnung (Key)	Wert (Value)
`$tp["Montag"]`	`Montag`	`17.5`
`$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`

Tabelle 1.6 Assoziatives Feld

Anders als bei numerisch indizierten Feldern sollten Sie die Ausgabe von einzelnen Elementen assoziativer Felder nicht innerhalb von Zeichenketten vornehmen. Das ist zwar möglich, führt aber häufig zu Fehlinterpretationen und zu Fehlern. Empfohlen wird daher die folgende Schreibweise:

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

Die `foreach`-Schleife bietet eine Möglichkeit, alle Elemente eines assoziativen Felds auszugeben.

In der ersten Schleife sorgt `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 `foreach($tp as $wert)` dafür, dass bei jedem Schleifendurchlauf jeweils nur der Value jedes Elements in der Variablen `$wert` bereitgestellt wird. Dieser Wert wird zur Berechnung des Mittelwerts aller Feldelemente genutzt.

Wie bei allen Verzweigungen und Schleifen gilt: Sollen mehrere Anweisungen ausgeführt werden, müssen sie innerhalb von geschweiften Klammern in einem Anweisungsblock notiert werden.

Innerhalb einer `foreach`-Schleife wird jeweils nur mit einer Kopie eines Feldelements gearbeitet. Eine Veränderung dieser Kopie hat keine Auswirkung auf die Inhalte des Felds. Mehr dazu lesen Sie in Abschnitt 8.7.

[»] **Hinweis**

Ordnen Sie einem bestimmten Schlüssel bei der Erzeugung des Felds oder später einen neuen Wert zu, wird nicht etwa ein neues Element hinzugefügt, sondern der erste Wert überschrieben. Die folgende Anweisung erzeugt also nur die beiden Feldelemente mit den Keys `Montag` und `Dienstag` und den Values `21.8` und `19.2`:

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

[✎] **Übung »u_feld_assoziativ«**

Es sollen der Vorname und das Alter von sechs Personen in einem assoziativen Feld gespeichert werden. Die Vornamen sollen die Keys und die Altersangaben die Values darstellen. Key und Value der Elemente des Felds sollen paarweise als Tabelle auf dem Bildschirm ausgegeben werden (Datei *u_feld_assoziativ.php*, siehe Abbildung 1.73).

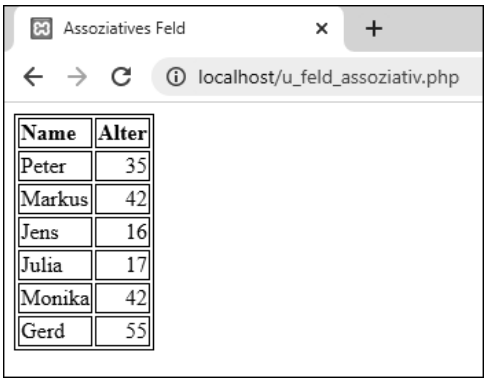


Abbildung 1.73 Ergebnis der Übung »u_feld_assoziativ«

Auch für die Erzeugung von assoziativen Feldern gibt es eine alternative Schreibweise. Die erste Zeile des oben angegebenen Programms *feld_assoziativ.php* hätte wie folgt lauten können:

```
$tp = ["Montag"=>17.5, "Dienstag"=>19.2, "Mittwoch"=>21.8];
```

Die Werte werden innerhalb von eckigen Klammern aufgelistet und der Variablen `$tp` zugewiesen. Auch damit wird `$tp`, mit derselben Indizierung, zu einem Feld.

1.10 Mehr über Funktionen

Nachdem Sie die Grundlagen zum Thema »Funktionen« kennengelernt haben, erläutere ich in diesem Abschnitt einige weitergehende Möglichkeiten. Sie können diesen Abschnitt zunächst auch überspringen und unmittelbar mit Abschnitt 1.11 fortfahren.

1.10.1 Variable Parameteranzahl

Es gibt vorgefertigte Funktionen, die es Ihnen ermöglichen, eine eigene Funktion mit unterschiedlichen Anzahlen von Parametern aufzurufen. Das erhöht die Flexibilität der eigenen Funktion, allerdings auch den Programmieraufwand.

Zu diesem Zweck stehen zur Verfügung:

- die Funktion `func_num_args()`, die die Anzahl der übergebenen Parameter ermittelt
- die Funktion `func_get_arg()`, die einen bestimmten Parameter aus der Parameterliste liefert

- die Funktion `func_get_args()`, die ein numerisch indiziertes Feld mit allen übergebenen Parametern liefert

Das folgende Programm verdeutlicht den Einsatz der beiden erstgenannten Funktionen:

```
<!DOCTYPE html>...<body>
<?php
   function addiere()
   {
      $anz = func_num_args();
      echo "<p>Anzahl der Werte: $anz<br>";
      echo "Werte: ";

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

Listing 1.51 Die Datei »funktion_get_arg.php«

Die Funktion `addiere()` wird insgesamt dreimal aufgerufen, jedes Mal mit einer anderen Anzahl an Parametern. Diese Anzahl wird mithilfe von `func_num_args()` ermittelt. Sie wird zur Steuerung einer `for`-Schleife verwendet.

Innerhalb der `for`-Schleife werden alle gelieferten Parameter mithilfe von `func_get_arg()` ausgegeben und addiert. Nach Beendigung der Schleife wird die Summe der Werte so wie in Abbildung 1.74 ausgegeben.

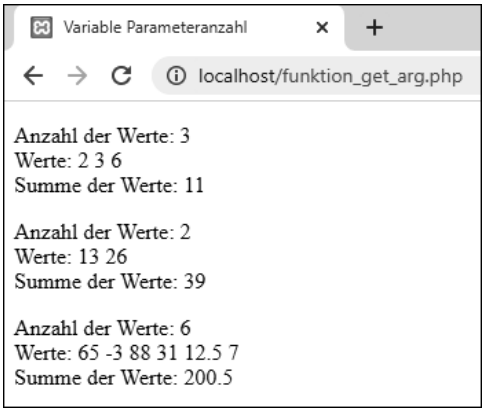


Abbildung 1.74 Variable Parameterlisten mit »func_get_arg()«

Eine alternative Lösung mithilfe der Funktion `func_get_args()` bietet das folgende Programm. Die Ausgabe entspricht der des vorherigen Beispiels:

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

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

Listing 1.52 Die Datei »funktion_get_args.php«

Mithilfe 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` ausgegeben und addiert.

1.10.2 Variadische Funktionen

Die *variadischen Funktionen* bieten eine weitere Möglichkeit, eine Funktion mit unterschiedlichen Anzahlen von Parametern aufzurufen. Das folgende Programm zeigt ein Beispiel. Die Ausgabe sieht genauso aus wie die Ausgabe der beiden Programme in Abschnitt 1.10.1:

```
<!DOCTYPE html>...<body>
<?php
   function addiere($eins, $zwei, ...$rest)
   {
      $anz = count($rest);
      echo "<p>Anzahl der Werte: " . (2 + $anz) . "<br>";
      echo "Werte: ";

      $sum = $eins + $zwei;
      $ausgabe = "$eins $zwei ";
      for($i=0; $i<$anz; $i++)
      {
         $sum = $sum + $rest[$i];
         $ausgabe .= "$rest[$i] ";
      }
      echo "$ausgabe<br>Summe der Werte: $sum</p>";
   }

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

Listing 1.53 Die Datei »funktion_variadisch.php«

Als Erstes folgt ein wichtiger Unterschied zu den beiden vorherigen Beispielen: Die Funktion `addiere()` muss diesmal mit mindestens zwei Parametern aufgerufen werden. Diese werden an die beiden Variablen `$eins` und `$zwei` übergeben. Weitere Parameter

werden im Feld `$rest` gespeichert und mithilfe der `for`-Schleife in die Rechnung einbezogen. Der Name des Felds muss dabei nach dem *Spread-Operator* notiert werden. Dieser besteht aus drei Punkten ....

Die Funktion `count()` wird häufig eingesetzt. Sie ermittelt die Anzahl der Elemente eines Felds. Allgemein ist die Funktion in der Lage, etwas in einem Objekt, hier in einem Feld, zu zählen. Sie kann auf alle Objekte angewendet werden, die das Interface `Countable` umsetzen. Mithilfe der Funktion `is_countable()`, die es seit PHP 7.3 gibt, können Sie feststellen, ob ein Objekt diese Schnittstelle umsetzt. Mehr zu Schnittstellen finden Sie in Abschnitt 4.12.

Für die Anzahl der Mindestparameter gibt es keine Vorgabe. Sie könnten also auch eine Funktion ohne Mindestparameter wie folgt definieren:

```
function addiere(...$rest) { [Code der Funktion] }
```

1.10.3 Parameter entpacken

Sie können Parameter einer Funktion auch »verpackt« übergeben. Anschließend werden sie in der Funktion »entpackt«. Betrachten Sie dazu das folgende Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   function mittelwert($a, $b, $c, $d, $e)
   {
      return ($a + $b + $c + $d + $e) / 5;
   }

   echo mittelwert(3.2, 14.5, 5.7, 4.2, 0.2) . "<br>";

   $feldEins = array(5.7, 4.2, 0.2);
   echo mittelwert(3.2, 14.5, ...$feldEins) . "<br>";

   $feldZwei = array(3.2, 14.5);
   echo mittelwert(...$feldZwei, ...$feldEins);
?>
</body></html>
```

Listing 1.54 Die Datei »funktion_entpacken.php«

Die Funktion `mittelwert()` berechnet den arithmetischen Mittelwert von fünf Werten, die ihr übergeben werden:

- Der erste Aufruf der Funktion erfolgt auf herkömmliche Art und Weise mithilfe von fünf einzelnen Zahlenwerten.
- Beim zweiten Aufruf werden zwei Einzelwerte und ein Feld mit drei Elementen übergeben. Vor dem Feldnamen steht der *Spread-Operator*. Dank des Operators wird das Feld innerhalb der Funktion entpackt und die Elemente werden den Parametern übergeben.
- Beim dritten Aufruf wird ein Feld mit zwei Elementen und ein Feld mit drei Elementen übergeben. Vor den Feldnamen steht jeweils der Spread-Operator. Beide Felder werden innerhalb der Funktion entpackt und die Elemente werden den Parametern übergeben.

Das Ergebnis für die Aufrufe sehen Sie in Abbildung 1.75.

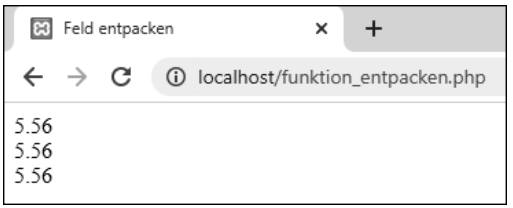


Abbildung 1.75 Verarbeitung von entpackten Werten

Beim Aufruf dürfen nach einem Feld, das entpackt wird, keine einzelnen Parameter mehr stehen. Beinhalten die Felder beim Aufruf der Funktion mehr Elemente als notwendig, werden die überzähligen Elemente ignoriert. Beinhalten sie weniger Elemente als notwendig, tritt ein Fehler auf, da die Funktion mit zu wenigen Parametern aufgerufen wird.

Seit PHP 7.4 können Felder auch zum Erzeugen neuer Felder entpackt werden (siehe Abschnitt 8.1).

1.10.4 Optionale Parameter

Sie können auch mit optionalen Parametern arbeiten. Für diese Parameter geben Sie Werte vor. Werden beim Aufruf der Funktion für diese Parameter keine Werte übergeben, werden die Vorgabewerte genommen. Dazu ein Beispiel:

```
<!DOCTYPE html>...<body>
<?php
   function volumen($laenge, $breite=1, $hoehe=1)
   {
```

```
    return $laenge * $breite * $hoehe;
}

echo "Volumen: " . volumen(2, 4, 0.6) . "<br>";
echo "Volumen: " . volumen(3.5, 2) . "<br>";
echo "Volumen: " . volumen(5);
?>
</body></html>
```

Listing 1.55 Die Datei »funktion_optional.php«

Die Funktion `volumen()` berechnet das Volumen eines Quaders gemäß der Formel *Länge × Breite × Höhe*. Die Länge muss beim Aufruf in jedem Fall übergeben werden, da dieser Parameter nicht optional ist. Die beiden Parameter `$breite` und `$hoehe` sind dagegen optional: Sollte die Höhe beim Aufruf nicht angegeben werden, wird gemäß dem Vorgabewert eine Höhe von 1 angenommen. Dasselbe gilt für die Breite.

Optionale Parameter können nur von rechts her angegeben werden. Bei der angegebenen Reihenfolge ist es also nicht möglich, nur mit einem optionalen Parameter für die Länge zu arbeiten oder nur mit zwei optionalen Parametern für die Länge und die Breite. Die Ausgabe des Programms sehen Sie in Abbildung 1.76.

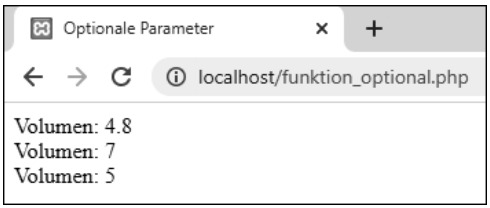


Abbildung 1.76 Nutzung von optionalen Parametern

Optionale Parameter können auch bei der Definition und dem Aufruf des Konstruktors oder anderer Methoden einer Klasse genutzt werden (siehe Abschnitt 4.6).

1.10.5 Benannte Parameter

Die Parameter einer Funktion werden normalerweise in der Reihenfolge aufgerufen, in der sie bei der Definition der Funktion notiert wurden. Diese Parameter werden auch als *positionale Parameter* bezeichnet.

Seit PHP 8.0 bietet PHP die Möglichkeit, mit *benannten Parametern* zu arbeiten. Das ergibt folgende Vorteile:

- Der Aufruf einer Funktion ist leichter lesbar, da die Namen der Parameter genannt werden.
- Sie können die Parameter in einer beliebigen Reihenfolge aufrufen.
- *Optionale Parameter* (siehe Abschnitt 1.10.4) können beim Aufruf übersprungen werden.

Im folgenden Programm werden zwei Funktionen zur Berechnung des Volumens eines Quaders auf unterschiedliche Arten aufgerufen:

```
<!DOCTYPE html>...<body>
<?php
   function volEins($lg, $br, $ho)
   {
      return $lg * $br * $ho;
   }
   echo "Volumen: " . volEins(3, 1.6, 4) . "<br>";
   echo "Volumen: " . volEins(br: 1.6, ho: 4, lg: 3) . "<br>";
   echo "Volumen: " . volEins(3, ho: 4, br: 1.6) . "<br><br>";

   function volZwei($lg, $br=1, $ho=1)
   {
      return $lg * $br * $ho;
   }
   echo "Volumen: " . volZwei(3, ho: 0.5) . "<br>";
   echo "Volumen: " . volZwei(ho: 0.5, lg: 3) . "<br>";
?>
</body></html>
```

Listing 1.56 Die Datei »funktion_benannt.php«

Bei einem Aufruf der Funktion `volEins()` müssen alle drei Parameter aufgeführt werden:

- Beim ersten Aufruf werden alle drei Parameter »klassisch« positional verwendet.
- Beim zweiten Aufruf erfolgt die Zuordnung ausschließlich über benannte Parameter. Der Name eines benannten Parameters wird ohne das vorangehende Zeichen `$` notiert, anschließend folgt das Zeichen `:` (Doppelpunkt), gefolgt vom Wert. Die Reihenfolge der benannten Parameter ist beliebig.
- Beim dritten Aufruf werden sowohl positionale als auch benannte Parameter genutzt.

Die Funktion `volZwei()` besitzt zwei optionale Parameter. Beim ersten Aufruf werden sowohl positionale als auch benannte Parameter genutzt, beim zweiten Aufruf nur benannte Parameter.

Achten Sie beim Aufruf einer Funktion mithilfe von benannten Parametern auf folgende Regeln:

- Nach einem benannten Parameter darf kein positionaler Parameter mehr folgen.
- Ein Parameter darf nur einmal erscheinen. Sie können also keinen der Parameter sowohl positional als auch benannt notieren.

Die Ausgabe des Programms sehen Sie in Abbildung 1.77.

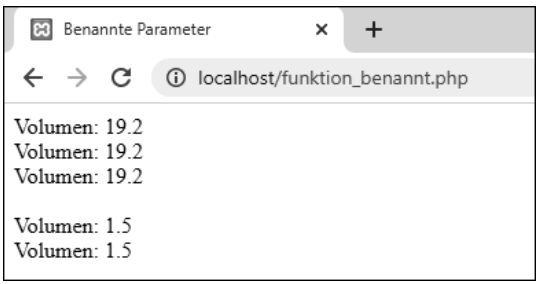


Abbildung 1.77 Einsatz von benannten Parametern

Benannte Parameter können auch bei der Definition und dem Aufruf des Konstruktors oder anderer Methoden einer Klasse genutzt werden (siehe Abschnitt 4.5).

1.10.6 Rekursive Funktionen

Funktionen können jederzeit andere Funktionen aufrufen. Man spricht hier von verschachtelten Aufrufen. Das Programm kehrt jeweils – aus einer beliebigen *Schachtelungstiefe* – zur aufrufenden Stelle zurück.

Funktionen können sich auch selbst aufrufen. Dieser Vorgang wird als *Rekursion* bezeichnet. Eine rekursive Funktion muss eine Verzweigung beinhalten, die die Rekursion wieder beendet, da es sonst zu einer endlosen Kette von Selbstaufrufen kommt. Bestimmte Problemstellungen lösen Sie programmiertechnisch am elegantesten durch eine Rekursion.

Im folgenden Programm wird eine Zahl so lange halbiert, bis ein bestimmter Grenzwert erreicht oder unterschritten wird. Zur Verdeutlichung wird der Halbierungsvorgang einmal mithilfe einer Schleife und einmal mithilfe einer Rekursion durchgeführt. Die Ausgabe des Programms sehen Sie in Abbildung 1.78.

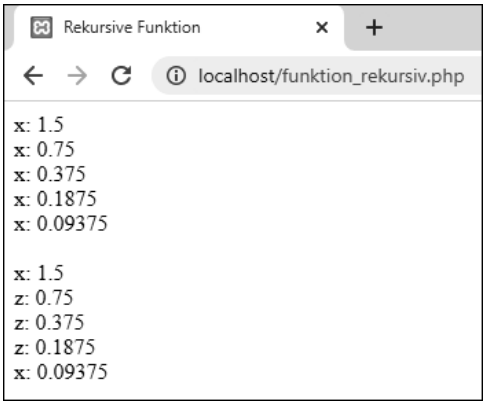


Abbildung 1.78 Halbierung mit Schleife bzw. mit Rekursion

Das Programm:

```
<!DOCTYPE html>...<body>
<?php
   /* Schleife */
   $x = 1.5;
   echo "x: $x<br>";
   while($x > 0.1)
   {
      $x = $x / 2;
      echo "x: $x<br>";
   }
   echo "<br>";

   /* Rekursion */
   function halbieren(&$z)
   {
      $z = $z / 2;
      if($z > 0.1)
      {
         echo "z: $z<br>";
         halbieren($z);
      }
   }

   $x = 1.5;
   echo "x: $x<br>";
```

```
    halbieren($x);
    echo "x: $x<br>";
?>
</body></html>
```

Listing 1.57 Die Datei »funktion_rekursiv.php«

Im ersten Teil des Programms wird die Variable `$x` in einer `while`-Schleife so lange halbiert, bis sie den Wert 0,1 erreicht oder unterschritten hat. Bei jedem Durchlauf der Schleife wird der aktuelle Wert angezeigt, sodass Sie die fortlaufende Halbierung verfolgen können.

Im zweiten Teil des Programms wird die Variable `$x` mithilfe der Funktion `halbieren()` halbiert. Anschließend wird geprüft, ob der Grenzwert erreicht oder unterschritten wird:

- Ist der Grenzwert noch nicht erreicht, ruft sich die Methode `halbieren()` selbst wieder auf. Dieser Vorgang kann sich mehrmals wiederholen.
- Ist der Grenzwert erreicht oder unterschritten, endet die Methode `halbieren()`. Gegebenenfalls endet sie damit mehrmals nacheinander. Das Programm endet mit der Ausgabe des Ergebnisses.

Die Variable `$x` (innerhalb der Methode heißt sie `$z`) wird jeweils per Referenz übergeben, daher wird immer die Originalvariable `$x` halbiert. Das können Sie an der letzten Ausgabe erkennen.

Ein weiteres Beispiel für eine rekursive Funktion steht in Abschnitt 7.8.

1.10.7 Auslagern von Funktionen

PHP-Code, den Sie in mehreren Programmen nutzen möchten, können Sie in externe Dateien auslagern. Dabei handelt es sich meist um eigene Funktionen. Es können aber auch einzelne Anweisungen ausgelagert werden.

Mithilfe der Anweisung `include` wird der Inhalt einer externen Datei in dasjenige Programm eingebunden, das ihn benötigt. Beachten Sie, dass der Code

- in der externen Datei in PHP-Markierungen sein muss,
- aber nicht in einem HTML-Dokument eingeschlossen sein sollte.

Geben Sie der externen Datei die Endung *.inc.php*. Am Kürzel *inc* erkennen Sie, dass der Inhalt in andere Dateien eingebunden wird. Mithilfe der Endung *php* sorgen Sie dafür, dass der Code nicht im Browser des Benutzers gelesen werden kann. Achten Sie auch bei dieser Datei darauf, dass sie UTF-8-kodiert ist (siehe Abschnitt 1.10.11).

Im folgenden Beispiel wird zunächst eine Funktion `maxi()` in der externen Datei *funktion_einbinden.inc.php* definiert. Die Funktion ermittelt aus den beiden übergebenen Parametern das Maximum, speichert diesen Wert in die Variable `$erg` und liefert ihn mithilfe der `return`-Anweisung zurück.

Die `return`-Anweisung steht im vorliegenden Fall innerhalb des `if`-Blocks beziehungsweise innerhalb des `else`-Blocks. Damit wird die Bearbeitung der Funktion unmittelbar unterbrochen, und der Programmablauf kehrt zur Aufrufstelle zurück:

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

Listing 1.58 Die Datei »funktion_einbinden.inc.php«

Die Funktion wird vom folgenden Programm aufgerufen. Dort wird zunächst die Datei *funktion_einbinden.inc.php* mithilfe der `include`-Anweisung eingebunden. Damit sind alle Funktionen aus der Datei *funktion_einbinden.inc.php* im aktuellen Programm bekannt und können verwendet werden.

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

Listing 1.59 Die Datei »funktion_einbinden.php«

Die Ausgabe des Programms sehen Sie in Abbildung 1.79.

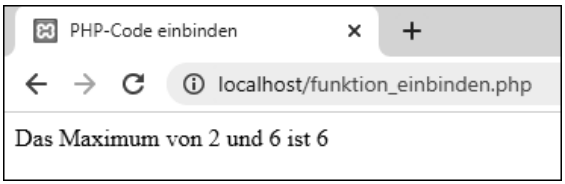


Abbildung 1.79 Nutzung einer eingebundenen Datei

Übung »u_funktion_einbinden« [✎]

Erstellen Sie eine kleine Funktionsbibliothek mit zwei Funktionen (Datei *u_funktion_einbinden.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. Es muss also die Summe dieser Zahlen durch ihre 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. Dazu ist die nachfolgend beschriebene 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 `$mx` sein, haben Sie ein neues Maximum gefunden, und dieser Wert wird `$mx` zugewiesen. Am Ende der Funktion wird `$mx` zurückgeliefert.

Testen Sie Ihre Bibliothek durch einige Aufrufe der beiden Funktionen mit unterschiedlich vielen Zahlen (Datei *u_funktion_einbinden.php*). Diese Bibliothek können Sie später erweitern und auch für andere Programme nutzen.

Externe Dateien lassen sich auch mit der Anweisung `require` statt mit der Anweisung `include` einbinden. Wird eine einzubindende Datei nicht gefunden, beendet `require` das Programm mit einem Fehler. Bei `include` wird lediglich eine Warnung ausgegeben, und das Programm läuft weiter.

Die Anweisungen `include_once` und `require_once` binden ebenfalls externe Dateien ein. Dabei wird darauf geachtet, dass eine einmal eingebundene Datei nicht ein zweites Mal eingebunden wird. In einer größeren Anwendung, die aus vielen Dateien besteht, ist es manchmal nicht leicht zu sehen, ob eine bestimmte Datei bereits eingebunden ist.

1.10.8 Anonyme Funktionen

Bisher wird mit *benannten Funktionen* gearbeitet. Diese Funktionen erhalten bei ihrer Definition einen eindeutigen Namen und werden über diesen Namen direkt aufgerufen.

Neben den benannten Funktionen gibt es die *anonymen Funktionen*. Sie erhalten bei ihrer Definition keinen Namen, ermöglichen Ihnen aber je nach Anwendungsfall eine flexiblere und kompaktere Programmierung.

Auf eine anonyme Funktion kann mithilfe einer Variablen verwiesen werden, die eine Verwendung der anonymen Funktion ermöglicht. Zudem können anonyme und benannte Funktionen als Parameter beim Aufruf einer anderen Funktion verwendet werden. (Dazu folgt mehr in Abschnitt 1.10.9.)

Hier folgt ein Programm, in dem zunächst eine anonyme Funktion ohne Parameter und ohne Rückgabewert definiert und genutzt wird. Anschließend wird mit einer anonymen Funktion mit zwei Parametern und einem Rückgabewert gearbeitet:

```
<!DOCTYPE html>...<body>
<?php
   $hallo = function()
   {
      echo "Hallo Welt<br>";
   };

   $hallo();

   $gruss = $hallo;
   $gruss();

   $summe = function($a, $b)
   {
      return $a + $b;
   };

   $addiere = $summe;
   echo "Addition: " . $addiere(3, 6.1);
?>
</body></html>
```

Listing 1.60 Die Datei »funktion_anonym.php«

In der ersten anonymen Funktion erfolgt nur die Ausgabe eines Texts. Auf die anonyme Funktion wird mit der Variablen `$hallo` verwiesen. Achten Sie darauf, den Verweis mit einem Semikolon nach der Definition der anonymen Funktion abzuschließen. Die anonyme Funktion wird über die Variable, gefolgt von runden Klammern, aufgerufen.

Sie können den Verweis einer weiteren Variablen zuweisen, hier `$gruss`. Anschließend ist ein Aufruf der anonymen Funktion über den zweiten Verweis möglich.

Die zweite anonyme Funktion erwartet zwei Parameter und liefert die Summe der beiden Werte zurück. Sie ist über die Variable `$summe` erreichbar. Auch diese anonyme Funktion wird einem zweiten Verweis zugewiesen, hier `$addiere`. Über diesen zweiten Verweis wird die anonyme Funktion anschließend aufgerufen.

Die Ausgabe des Programms sehen Sie in Abbildung 1.80.

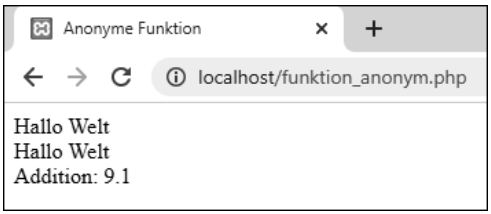


Abbildung 1.80 Anonyme Funktionen

1.10.9 Callback-Funktionen

Bei *Callback-Funktionen* (deutsch: *Rückruffunktionen*) handelt es sich um benannte oder anonyme Funktionen, die einer anderen Funktion als Parameter übergeben werden.

Im folgenden Programm wird der Funktion `ausgabe()` beim ersten Aufruf die benannte Funktion `haelfte()` als vierter Parameter übergeben, beim zweiten Aufruf eine anonyme Funktion:

```
<!DOCTYPE html> ... <body>
<?php
   function haelfte($x)
   {
      return $x / 2;
   }

   function ausgabe($von, $bis, $schritt, $fkt)
   {
```

```
        for($i=$von; $i<$bis; $i+=$schritt)
            echo $fkt($i) . " ";
        echo "<br>";
    }

    ausgabe(5, 14, 2, "haelfte");
    ausgabe(5, 14, 2, function($x) { return $x / 4; });
?>
</body></html>
```

Listing 1.61 Die Datei »funktion_callback.php«

Die eigene, benannte Funktion `ausgabe()` soll eine Reihe von Werten ausgeben. Der Funktion werden insgesamt vier Parameter übergeben. Die ersten drei Parameter sorgen dafür, dass eine Schleife mit den Werten von 5 bis 13 mit einer Schrittweite von 2 durchlaufen wird.

Bei dem vierten Parameter handelt es sich um die Variable `$fkt`, die auf eine Funktion verweist. Eine solche Funktion wird auch Callback-Funktion genannt. Die hier vorliegende Callback-Funktion erwartet einen Parameter und liefert einen Rückgabewert. In der Funktion `ausgabe()` wird die Callback-Funktion mehrmals aufgerufen. Bei jedem Aufruf ermittelt die Callback-Funktion einen Wert und liefert diesen zurück. In der Funktion `ausgabe()` wird dieser Wert ausgegeben.

Als Callback-Funktion können sowohl benannte als auch anonyme Funktionen dienen. In Abhängigkeit von der jeweiligen Callback-Funktion werden unterschiedliche Werte ermittelt, an die Funktion `ausgabe()` zurückgeliefert und dort ausgegeben.

Die Callback-Funktionen müssen so aufgebaut sein, wie es bei der Definition der Funktion `ausgabe()` vorgegeben wird: Sie müssen einen Parameter erwarten und einen Rückgabewert liefern.

Beim ersten Aufruf der Funktion `ausgabe()` wird die benannte Funktion `haelfte()` als Callback-Funktion übergeben. Sie liefert für jeden übergebenen Wert die Hälfte des Werts zurück.

Beim zweiten Aufruf der Funktion `ausgabe()` wird eine anonyme Funktion als Callback-Funktion übergeben. Sie liefert für jeden übergebenen Wert ein Viertel des Werts zurück.

Die Ausgabe des Programms sehen Sie in Abbildung 1.81.

Abbildung 1.81 Callback-Funktionen

1.10.10 Generatoren

Die *Generatoren* stellen einen besonderen Typ von Funktionen dar. Sie liefern nicht nur einen einzelnen Wert, sondern mehrere Werte.

Dabei steht zu jedem Zeitpunkt nur ein Wert im Arbeitsspeicher. Es müssen nicht alle Werte der Reihe gleichzeitig im Arbeitsspeicher aufbewahrt werden. Auf diese Weise wird die Nutzung des Arbeitsspeichers optimiert. Als Generator wird eine Funktion definiert, aus der die Werte mithilfe des Schlüsselworts `yield` geliefert werden.

Im Unterschied zu einer normalen Funktion, die nur einen Wert liefern kann, liefert eine Generatorfunktion mehrere Werte. Wird sie innerhalb einer `foreach`-Schleife aufgerufen, endet die Generatorfunktion nicht nach einem Aufruf, sondern wird beim nächsten Aufruf fortgesetzt. Auf diese Weise werden alle generierten Werte nacheinander geliefert.

Im folgenden Beispiel wird ein Generator definiert, der eine Reihe von Würfelwerten bereitstellt:

```
<!DOCTYPE html>...<body>
<?php
   function wuerfelwertGenerator()
   {
      for($i=1; $i<=10; $i++)
         yield random_int(1,6);
   }
   foreach(wuerfelwertGenerator() as $wert)
      echo "$wert ";
?>
</body></html>
```

Listing 1.62 Die Datei »funktion_generator.php«

Der Generator `wuerfelwertGenerator()` generiert insgesamt zehn zufällige Werte zwischen 1 und 6. Die folgende `foreach`-Schleife wird mit diesem Generator aufgerufen. Bei

jedem Durchlauf der `foreach`-Schleife wird mithilfe von `yield` ein einzelner, zufälliger Wert bereitgestellt. Die Ausgabe des Programms sehen Sie in Abbildung 1.82.

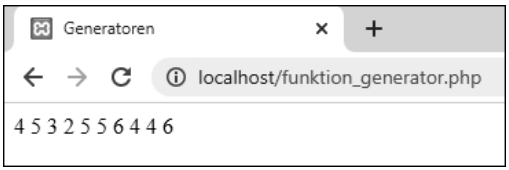


Abbildung 1.82 Generator für Würfelwerte

1.10.11 Typhinweise

Seit PHP 7.0 wird Ihnen mithilfe von Typhinweisen die Möglichkeit geboten, Datentypen stärker zu kontrollieren. Diese Kontrolle bezieht sich auf die Parameter und den Rückgabewert von Funktionen. Dabei kann es sich um eigene Funktionen oder auch um vordefinierte Funktionen handeln.

Zunächst ein Beispiel:

```
<?php declare(strict_types=1); ?>
<!DOCTYPE html>...<body>
<?php
   echo "Typ int:<br>";
   function addiere(int $a, int $b):int
   {
      $c = $a + $b;
      return $c;
      // return $c * 1.0;
   }
   echo addiere(1, 2) . "<br>";
   // echo addiere(1.9, 2.9) . "<br>";
   echo "Ende des Programms";
?>
</body></html>
```

Listing 1.63 Die Datei »funktion_typhinweise.php«

Soll sich die Kontrolle der Datentypen innerhalb einer Datei auswirken, muss die Anweisung `declare(strict_types=1)`; als allererste Anweisung der Datei notiert werden, wie es im Beispiel zu sehen ist.

Enthält eine PHP-Datei zu Beginn eine `declare`-Anweisung, müssen Sie darauf achten, dass die Kodierung der Datei auf dem Wert *UTF-8* steht. Sie können sie im Menü KODIERUNG des Editors Notepad++ überprüfen und bei Bedarf korrigieren, und zwar über den Menüpunkt KONVERTIERE ZU UTF-8.

Im Programm wird die Funktion `addiere()` definiert. Sie dient zur Addition von zwei Zahlenwerten und zur Rückgabe des Ergebnisses an die aufrufende Stelle. Die Ausgabe des Programms sehen Sie in Abbildung 1.83.

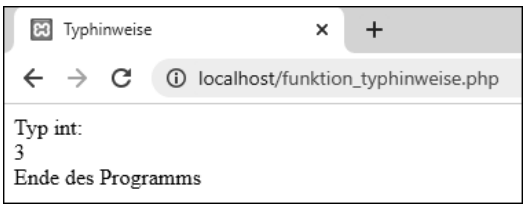


Abbildung 1.83 Typhinweise für »int«

Das Neue an diesem Programm ist:

- Die beiden Parameter der Funktion müssen ganzzahlig sein. Zu diesem Zweck wird der Typhinweis `int` vor jedem Parameter notiert.
- Der Rückgabewert der Funktion muss ebenfalls ganzzahlig sein. Dazu werden nach den Parameterklammern ein Doppelpunkt und wiederum der Typhinweis `int` notiert.

Der Funktionsaufruf `addiere(1, 2);` erfüllt die genannten Bedingungen und kann daher durchgeführt werden. Der Funktionsaufruf `addiere(1.9, 2.9);` erfüllt die Bedingungen nicht, weil die Parameter nicht ganzzahlig sind. Die Anweisung `return $c * 1.0;` erfüllt die Bedingungen ebenfalls nicht, weil der Rückgabewert nicht ganzzahlig ist. Sind die Bedingungen nicht erfüllt, folgt eine Fehlermeldung und die letzte Ausgabe mit dem Text »Ende des Programms« ist nicht mehr zu sehen.

Eine Kontrolle dieser Art trägt zur Verbesserung der Lesbarkeit, des Ablaufs und der Pflege Ihrer Programme bei. Zunächst arbeiten Sie mit den folgenden Typhinweisen:

- `int`: für ganze Zahlen
- `float`: für Fließkommazahlen
- `string`: für Zeichenketten
- `bool`: für Wahrheitswerte

Es gibt noch weitere Typhinweise, zum Beispiel für Objekte (siehe Abschnitt 4.9) und Felder (siehe Abschnitt 8.6).

Die Kontrolle mithilfe der `declare`-Anweisung gilt bei Parametern für die Datei, in der die Funktion aufgerufen wird. Bezüglich des Rückgabewerts gilt sie für die Datei, in der die Funktion definiert wird.

Es findet keine automatische Umwandlung statt. Ein Beispiel: Fließkommazahlen oder Zeichenketten, die Zahlen enthalten, werden nicht einfach in ganze Zahlen konvertiert. Es gibt aber keine Regel ohne Ausnahme: Bei einem `float`-Parameter oder bei einem Rückgabewert des Typs `float` werden auch ganze Zahlen akzeptiert.

Programme mit Typhinweisen und der `declare`-Anweisung können nicht in Versionen vor PHP 7.0 genutzt werden.

Ein weiteres Beispiel:

```
<?php declare(strict_types=1); ?>
<!DOCTYPE html>...<body>
<?php
   echo "Typ float: <br>";
   function addiereFloat(float $a, float $b):float
   {
      $c = $a + $b;
      return $c;
   }
   echo addiereFloat(1.9, 2.9) . "<br>";
   echo addiereFloat(1, 2) . "<br>";
   // echo addiereFloat("1.9", 2.9) . "<br>";

   echo "Typ bool: ";
   function oder(bool $a, bool $b):bool
   {
      $c = $a || $b;
      return $c;
   }
   echo oder(true, false) . "<br>";
   // echo oder(1, "abc") . "<br>";

   echo "Typ string: ";
   function verkette(string $a, string $b):string
   {
      $c = $a . $b;
      return $c;
   }
```

```
    echo verkette("Hallo", "Welt") . "<br>";
    // echo verkette(5, 8.2) . "<br>";

    echo "Vordefinierte Funktionen: ";
    echo mb_strlen("Hallo") . "<br>";
    // echo mb_strlen(123) . "<br>";
    echo "Ende des Programms";
?>
</body></html>
```

Listing 1.64 Die Datei »funktion_typhinweise_weitere.php«

Die Ausgabe des Programms sehen Sie in Abbildung 1.84.

Abbildung 1.84 Typhinweise für weitere Datentypen

Die Funktion `addiereFloat()` erwartet zwei `float`-Werte, addiert sie und liefert einen `float`-Wert zurück. Die ersten beiden Aufrufe der Funktion erfüllen die Bedingungen. Im zweiten Fall werden die beiden ganzen Zahlen zu `float`-Werten erweitert. Der dritte Aufruf erfüllt die Bedingungen nicht.

Die Funktion `oder()` erwartet zwei `bool`-Werte, ermittelt einen `bool`-Wert als Ergebnis und liefert diesen zurück. Nur der erste Aufruf der Funktion erfüllt die Bedingungen.

Die Funktion `verkette()` erwartet zwei `string`-Werte, ermittelt einen `string`-Wert als Ergebnis und liefert diesen zurück. Auch hier erfüllt nur der erste Aufruf der Funktion die Bedingungen.

Die vordefinierte Funktion `mb_strlen()` ermittelt die Anzahl der Zeichen einer Zeichenkette. Sie erwartet einen `string`-Wert. Auch hier erfüllt nur der erste Aufruf der Funktion die Bedingungen. Mehr zu den vordefinierten Funktionen für Zeichenketten finden Sie in Kapitel 6.

In den vorliegenden Beispielen sind die Datentypen der Parameter und der Datentyp des Rückgabewerts stets gleich. Sie können aber auch unterschiedlich sein. Die genannte Kontrolle kann sich also zum Beispiel auf eine Funktion beziehen, die zwei `int`-Parameter, einen `float`-Parameter und einen `string`-Parameter erwartet und einen `bool`-Wert oder auch gar keinen Wert als Ergebnis zurückliefert.

1.10.12 Nullbare Typen

Mit PHP 7.1 wurden die *nullbaren Typen* (englisch: *nullables*) eingeführt. Zur Erzeugung eines nullbaren Typs können Sie dem Typhinweis einer Variablen das Zeichen `?` voranstellen. In diesem Fall darf die Variable

- entweder einen Wert haben, der zum betreffenden Typhinweis passt,
- oder den Wert `null` haben.

Viele vordefinierte Funktionen liefern

- entweder den ermittelten Rückgabewert zurück
- oder den Wert `null` als Information dafür, dass innerhalb der Funktion ein Fehler aufgetreten ist.

Arbeitet man ohne Typhinweise, lässt sich dieses Verhalten auch für eigene Funktionen erzeugen, da der Typ des Rückgabewerts beliebig ist.

Möchte man allerdings die Vorteile von Typhinweisen nutzen, so lässt sich dieses Verhalten für eigene Funktionen nur mithilfe von nullbaren Typen erzeugen. Es folgt ein Beispiel:

```
<?php declare(strict_types=1); ?>
<!DOCTYPE html>...<body>
<?php
   echo "Nullable-Typ:<br>";
   function dividiere(float $a, float $b):?float
   {
      if($b == 0)
         return null;
      else
         return $a / $b;
   }

   $ergebnis = dividiere(8, 5);
   // $ergebnis = dividiere(8, 0);
```

```
   if(isset($ergebnis))
      echo $ergebnis . "<br>";
   else
      echo "Fehler in Funktion<br>";
   echo "Ende des Programms";
?>
</body></html>
```

Listing 1.65 Die Datei »funktion_rueckgabewert_nullbar.php«

Die eigene Funktion `dividiere()` soll entweder das Ergebnis der Division als einen Wert des Typs `float` liefern (siehe Abbildung 1.85) oder den Wert `null` als Information dafür, dass versucht wurde, eine Division durch `0` durchzuführen (siehe Abbildung 1.86). Zu diesem Zweck hat der Rückgabewert der Funktion den Typhinweis `?float`.

Der Rückgabewert wird in der Variablen `$ergebnis` gespeichert. Im Programm kann mithilfe der Funktion `isset()` geprüft werden, ob bei der Funktion ein Fehler aufgetreten ist.

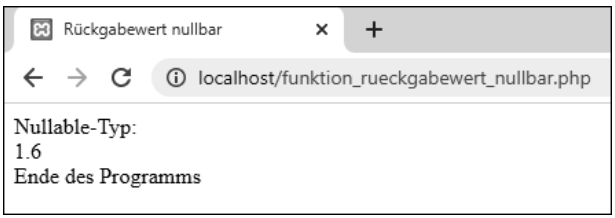


Abbildung 1.85 Rückgabe eines »float«-Werts

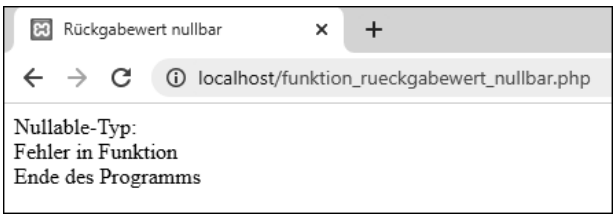


Abbildung 1.86 Rückgabe von »null«

1.11 Beispiele

In diesem Abschnitt werden einige umfangreichere Beispiele vorgestellt. Sie beinhalten keine neuen Programmierelemente, sondern dienen nur zur Verdeutlichung des Zusammenspiels verschiedener Elemente. Die genaue Beschreibung erfolgt mithilfe von

Kommentaren im Code, und Sie finden die Programme in den Materialien zum Buch unter *www.rheinwerk-verlag.de/5618*.

Zunächst folgt aber ein kleiner Abschnitt über die sinnvolle Arbeitsweise zur Entwicklung eines größeren Programms.

1.11.1 Entwicklung eines Programms

Bei der Entwicklung Ihrer eigenen Programme sollten Sie Schritt für Schritt vorgehen. Stellen Sie zuerst einige Überlegungen darüber an, wie das gesamte Programm aufgebaut sein sollte, und zwar auf Papier. Aus welchen Teilen sollte es nacheinander bestehen? Versuchen Sie *nicht*, das gesamte Programm mit all seinen komplexen Bestandteilen auf einmal zu schreiben! Dies ist der größte Fehler, den Einsteiger und Einsteigerinnen (und manchmal auch Fortgeschrittene) machen können.

Schreiben Sie zunächst eine einfache Version des ersten Programmteils. Anschließend testen Sie sie. Erst nach einem erfolgreichen Test fügen Sie den folgenden Programmteil hinzu. Nach jeder Änderung testen Sie wiederum. Sollte sich ein Fehler zeigen, wissen Sie, dass er aufgrund der letzten Änderung aufgetreten ist. Nach dem letzten Hinzufügen haben Sie eine einfache Version Ihres gesamten Programms.

Nun ändern Sie einen Teil Ihres Programms in eine komplexere Version ab. Auf diese Weise machen Sie Ihr Programm Schritt für Schritt komplexer, bis Sie schließlich das gesamte Programm so erstellt haben, wie es Ihren anfänglichen Überlegungen auf Papier entspricht.

Manchmal ergibt sich während der praktischen Programmierung noch die eine oder andere Änderung gegenüber Ihrem Entwurf. Das ist kein Problem, solange sich nicht der gesamte Aufbau ändert. Sollte das allerdings der Fall sein, kehren Sie noch einmal kurz zum Papier zurück und überdenken Sie den Aufbau. Das bedeutet nicht, dass Sie die bisherigen Programmzeilen löschen müssen, aber möglicherweise müssen Sie sie ein wenig ändern und anders anordnen.

Schreiben Sie Ihre Programme übersichtlich. Falls Sie gerade überlegen, wie Sie drei, vier bestimmte Schritte Ihres Programms auf einmal machen können: Erstellen Sie daraus einfach einzelne Anweisungen, die der Reihe nach ausgeführt werden. Dies vereinfacht eine eventuelle Fehlersuche. Möchten Sie (oder andere Personen) Ihr Programm später einmal ändern oder erweitern, gelingt der Einstieg in den Aufbau des Programms wesentlich schneller.

Eine typische Frage, die Sie sich besonders im Zusammenhang mit Verzweigungen und Schleifen immer wieder stellen sollten, lautet: Haben Sie alle Klammern, die Sie geöffnet haben, auch wieder geschlossen?

Sie können die Anweisung `echo` auch zur Kontrolle von Werten und zur Suche von logischen Fehlern einsetzen. Außerdem können Sie einzelne Teile Ihres Programms in Kommentarklammern setzen, um festzustellen, welcher Teil des Programms fehlerfrei läuft und welcher Teil demnach fehlerbehaftet ist.

1.11.2 Geldanlage

Ein Benutzer besucht die Website einer Bank, die verschiedene Möglichkeiten zur Geldanlage bietet. Eine dieser Möglichkeiten ist die Anlage eines bestimmten Betrags über eine festgelegte Laufzeit. Je länger das Geld angelegt wird, desto höher ist der Zinssatz. Der Benutzer gibt in einem Formular den angelegten Betrag sowie die Laufzeit ein (siehe Abbildung 1.87).

Abbildung 1.87 Eingabeformular »Geldanlage«

Er erhält als Antwort eine Tabelle, in der die Entwicklung seiner Geldanlage von Jahr zu Jahr dargestellt wird (siehe Abbildung 1.88).

Die genaue Beschreibung erfolgt mithilfe von Kommentaren im Code der beiden Dateien *geldanlage.htm* und *geldanlage.php*.

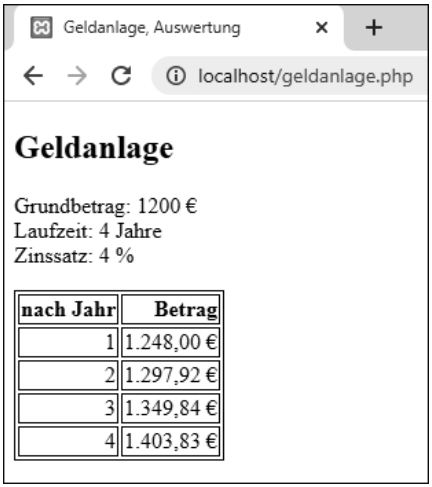


Abbildung 1.88 Ausgabe »Geldanlage«

1.11.3 Steuertabelle

Es soll eine (stark vereinfachte) Berechnung und Ausgabe von Steuersätzen und Steuerbeträgen vorgenommen werden. Der Steuersatz wird abhängig vom Einkommen nach Tabelle 1.7 berechnet. Die Benutzerin gibt im Formular (siehe Abbildung 1.89) die folgenden Daten ein:

- **Startwert:** erster Wert, für den das Ergebnis berechnet wird
- **Endwert:** letzter Wert, für den das Ergebnis berechnet wird
- **Intervall:** Abstand der einzelnen Werte voneinander

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

Tabelle 1.7 Einkommen und Steuersätze

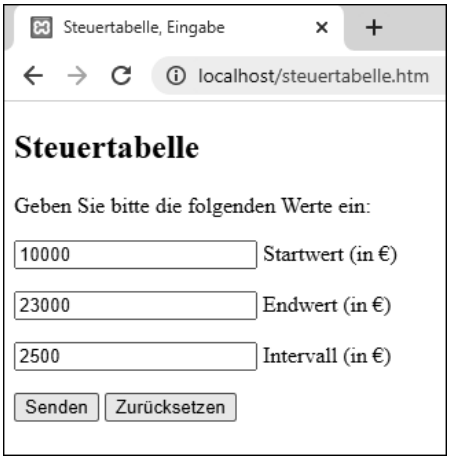
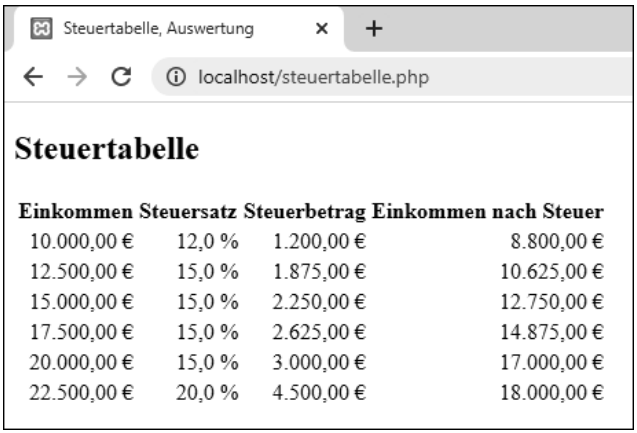


Abbildung 1.89 Eingabeformular »Steuertabelle«

Die Ausgabe zu den Beispieldaten sehen Sie in Abbildung 1.90.



Einkommen	Steuersatz	Steuerbetrag	Einkommen nach Steuer
10.000,00 €	12,0 %	1.200,00 €	8.800,00 €
12.500,00 €	15,0 %	1.875,00 €	10.625,00 €
15.000,00 €	15,0 %	2.250,00 €	12.750,00 €
17.500,00 €	15,0 %	2.625,00 €	14.875,00 €
20.000,00 €	15,0 %	3.000,00 €	17.000,00 €
22.500,00 €	20,0 %	4.500,00 €	18.000,00 €

Abbildung 1.90 Ausgabe »Steuertabelle«

Die genaue Beschreibung erfolgt mithilfe von Kommentaren im Code der beiden Dateien *steuertabelle.htm* und *steuertabelle.php*.

1.11.4 Bestimmung des Ostersonntags

In diesem Abschnitt wird die Funktion `ostersonntag()` zur Bestimmung des Termins des Ostersonntags in einem vorgegebenen Jahr entwickelt. Auf Basis des Ostersonntags

können alle beweglichen Feiertage eines Bundeslandes berechnet werden. In Abschnitt 9.9 wird auf diese Weise eine Liste der (beweglichen und festen) Feiertage erstellt. Die Funktion `ostersonntag()` wird in die eingebundene Datei *ostersonntag.inc.php* ausgelagert.

Der Benutzer gibt im Formular zwei Jahreszahlen ein. Das Programm liefert eine Tabelle, in der zu jedem Jahr im angegebenen Jahresbereich der jeweilige Termin des Ostersonntags ausgegeben wird. Nimmt der Benutzer eine Eingabe vor wie in Abbildung 1.91, ...

Abbildung 1.91 Eingabe eines Jahresbereichs

... wird die Tabelle aus Abbildung 1.92 geliefert.

Abbildung 1.92 Ostersonntage im eingegebenen Bereich

Ostern ist stets am ersten Sonntag nach dem ersten Vollmond des Frühlings. So hat es das erste Kirchenkonzil im Jahr 325 n. Chr. festgelegt, und dies gilt bis heute. Im Jahr 1800 entwickelte der deutsche Mathematiker Carl Friedrich Gauß (1777–1855) eine Formel zur Berechnung des Ostersonntags, die in dieser Anwendung genutzt wird. Sie ist so genau, dass erst für das Jahr 8202 ein Fehler auftritt.

Die genaue Beschreibung erfolgt mithilfe von Kommentaren im Code der beiden Dateien *ostersonntag.htm* und *ostersonntag.php* und der eingebundenen Datei *ostersonntag.inc.php*.