Javascript und php

Client- und serverseitige Anwendungsentwicklung für das WWW

Bernd Blümel

Version: 29. April 2002

Inhaltsverzeichnis

1 Einleitung **2**

2 Erste Beispiele **6**

3 Einfügen von JavaScript und php in html-Dateien **25**
3.1 Einfügen von JavaScript in html-Dateien . . . 25
3.1.1 <script> und < /script> . . . 25
3.1.2 Das src-Attribut des <script>-Tags . . . 25
3.2 Einfügen von php in html-Dateien . . . 27

4 Grundsätzliches **30**
4.1 Kommentare . . . 30
4.2 Variablen . . . 31
4.3 Ausdrücke . . . 37
4.4 Operatoren . . . 37
4.4.1 Der Zuweisungsoperator . . . 37
4.4.2 Arithmetische Operatoren . . . 37
4.4.3 Vergleichsoperatoren . . . 41
4.4.4 Logische Operatoren . . . 43
4.4.5 Der tenäre Operator . . . 44
4.5 Konstante . . . 45

5 Beispiele **49**
5.1 Euro-Dollar-Umrechnung . . . 49
5.2 Volatilität . . . 51
5.3 Das Raketenbeispiel . . . 55

6 Algorithmen und Kontrollstrukturen (Steuerungsstrukturen) **64**
6.1 Algorithmen . . . 64
6.2 Die if- Anweisung (Ein- und Zweiseitige Auswahl) . . . 65
6.2.1 Motivation . . . 65
6.2.2 Syntax . . . 68
6.2.3 php und Get und Post . . . 73
6.2.4 Beispiele . . . 76
6.3 Der Switch (Mehrfachauswahl) . . . 89
6.3.1 Motivation . . . 89
6.3.2 Syntax . . . 91
6.3.3 Beispiele . . . 95

7 Funktionen **106**
7.1 Motivation . . . 106
7.2 Syntax . . . 112
7.3 Beispiele . . . 116
7.3.1 Euro-Dollar-Umrechnung (fortgesetzt) . . . 116
7.3.2 Volatilitäten (fortgesetzt) . . . 125
7.3.3 Raketenbeispiel (fortgesetzt) . . . 125
7.4 Referenz- und Wertparameter . . . 133

8 Klassen und Objekte **142**
8.1 Motivation . . . 142
8.2 Beispiele in php . . . 145
8.2.1 Euro-Dollar-Umrechnung objektorientiert . . . 145
8.2.2 Volatilitäten (objektorientiert) . . . 149
8.2.3 Raketenbeispiel (objektorientiert) und die Nutzung von Konstruktoren . . . 149
8.3 Klassenmethoden . . . 158
8.4 Klassen und Objekte in JavaScript . . . 163

9 JavaScript-Objekte und das Event-Modell **168**
9.1 Das Window-Objekt und die Objekthierarchie . . . 168
9.2 Das document-Objekt . . . 173
9.3 Event-Handler . . . 175
9.4 Das link-Objekt . . . 177
9.5 Das Form-Objekt . . . 182
9.5.1 Eingabekontrollen . . . 186

10 Das große Ganze: Teil 1 **203**
10.1 Aufgabenstellung und Analyse . . . 203
10.2 Einige Bemerkungen zur Nutzung von Datenbanksystemen in php . . . 205
10.3 Erste Lösung, Stringbehandlung . . . 208
10.4 Verbesserte Lösung: Sessions und ein wenig Anwendungsarchitektur . . . 224

11 Lösungen **248**

Kapitel 1

Einleitung

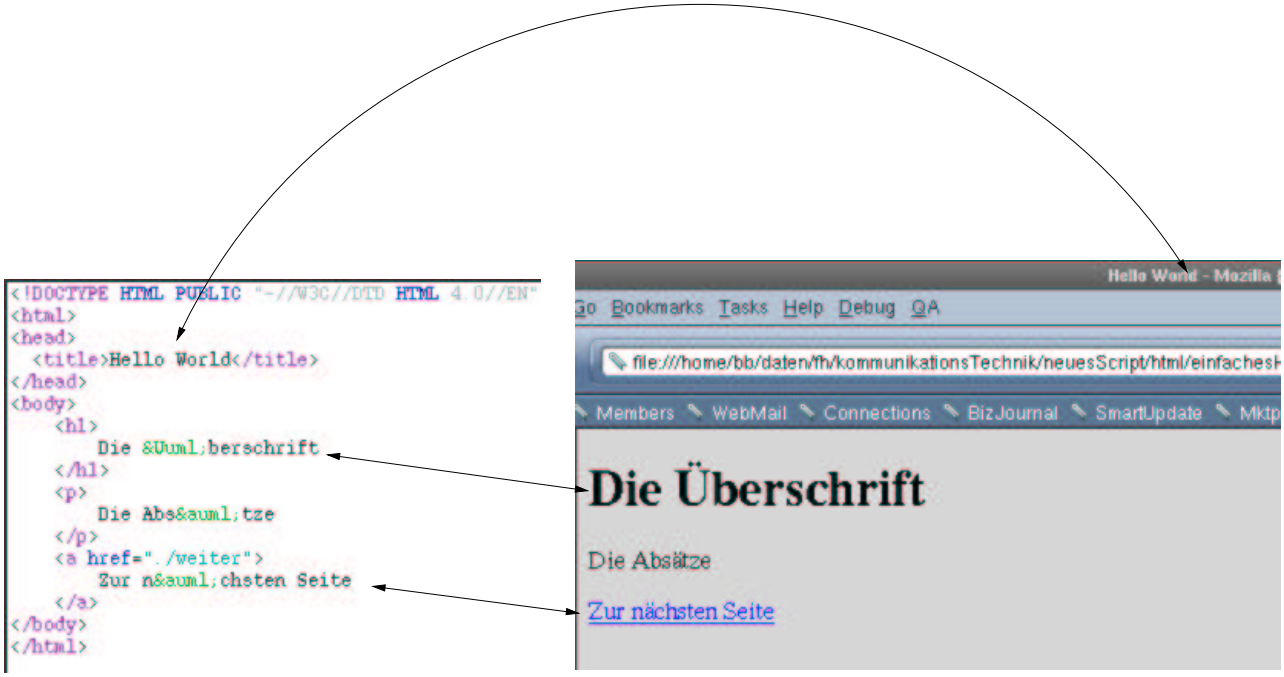


Abbildung 1.1: html-Code und Interpretation durch den Browser

In Abb. 1.1 sehen Sie eine einfache html-Datei und ihre Darstellung in einem Browser. Dieses html-Beispiel zeigt natürlich nicht den ganzen Funktionsumfang von html[1], wir können jedoch an diesem Beispiel bereits prinzipielle Einschränkungen von html erkennen:

- Eine wie in Abb. 1.1 dargestellte Seite ist nach dem Laden der Seite nicht mehr änderbar.
- Ganz prinzipiell steht es mit den Interaktionsmöglichkeiten in html nicht zum Besten: Die einzige vorgesehene Möglichkeit stellen Formulare dar. Abb. 1.2 zeigt ein Beispiel eines solchen Formulars. In html steht für die Bearbeitung der Eingaben eines solchen Formulars die CGI-Schnittstelle (Common Gateway Interface) zur Verfügung. Dazu gibt man bei der Definition

[1] Wäre sonst auch traurig, wo wir uns ja bereits einige Stunden mit html beschäftigt haben

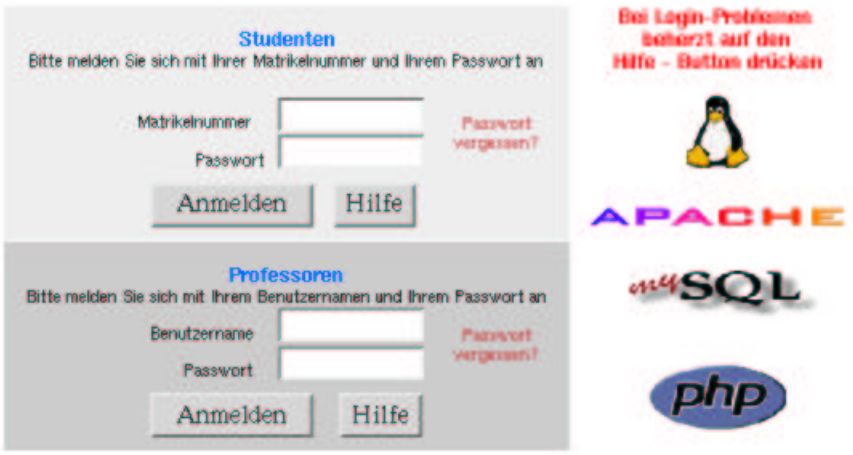


Abbildung 1.2: Anmeldungsseite zu unserem Intranet

des Formulars in der html-Quelle den Namen eines Programms an. Abb. 1.2 enthält zwei Formulare. Für den Professorenteil könnte dies die Zeile [2]

```
<form name= "profsendform" action="/cgi-bin/profLogin" method="post">
```

sein. Auf dem Server muß es ein Programm mit dem Namen profLogin geben. Wenn die Schaltfläche „Absenden“ ausgelöst wird, startet der Server dieses Programm und übergibt dem Programm die Inhalte der Eingabefelder. Das Programm kann nun beliebige Sachen tun – in unserem Fall sollte es überprüfen, ob ein Professor mit dem angegebenen Namen und Password existiert – und dann seine Ergebnisse an den WWW-Server zurückgeben. Der WWW-Server sendet die Ergebnisse zurück zum Browser. Dieser Vorgang ist zusammenfassend in Abb. 1.3 dargestellt.

- Clientseitig ist überhaupt keine Verarbeitung möglich. So wäre es bei der Anwendung in Abb. 1.2 sicher sinnvoll, zu überprüfen, ob der Benutzer überhaupt Eingaben gemacht hat. Und das am besten durch den Browser, denn dadurch könnte man vermeiden, dass ein leeres Formular überhaupt an den Server geschickt wird. Dies ist jedoch in html nicht möglich.

Um diesen (und anderen) Nachteilen abzuhelfen, werden ständig neue Technologien entwickelt. Diese Technologien können wir anhand ihres Einsatzortes unterscheiden. Sie dienen entweder dazu, clientseitige Interaktion zu ermöglichen, was bedeutet, dass sie im Browser implemetiert sein müssen, oder es sind serverseitige Technologien, was bedeutet, dass sie die CGI-Schnittstelle ersetzen.

Clientseitige Technologien sollen überhaupt erst Interaktion der Benutzer mit dem Browser ermöglichen. Beispiele clientseitiger Technologien sind u.a.:

- *Plug-Ins*: Plug-Ins sind Programme, die in den jeweiligen Browser integrierbar sind. Sie müssen entweder getrennt erworben werden – was im Normalfall „laden von dem Server des Anbieters“ bedeutet – oder sie werden bereits mit dem Browser ausgeliefert. Beispiele solcher Plug-Ins sind Flash, Real video oder der Acrobat Reader als Plug-In.

[2] Ist sie allerdings nicht, denn unser Intranet beruht auf JavaScript und php, doch das kommt später.

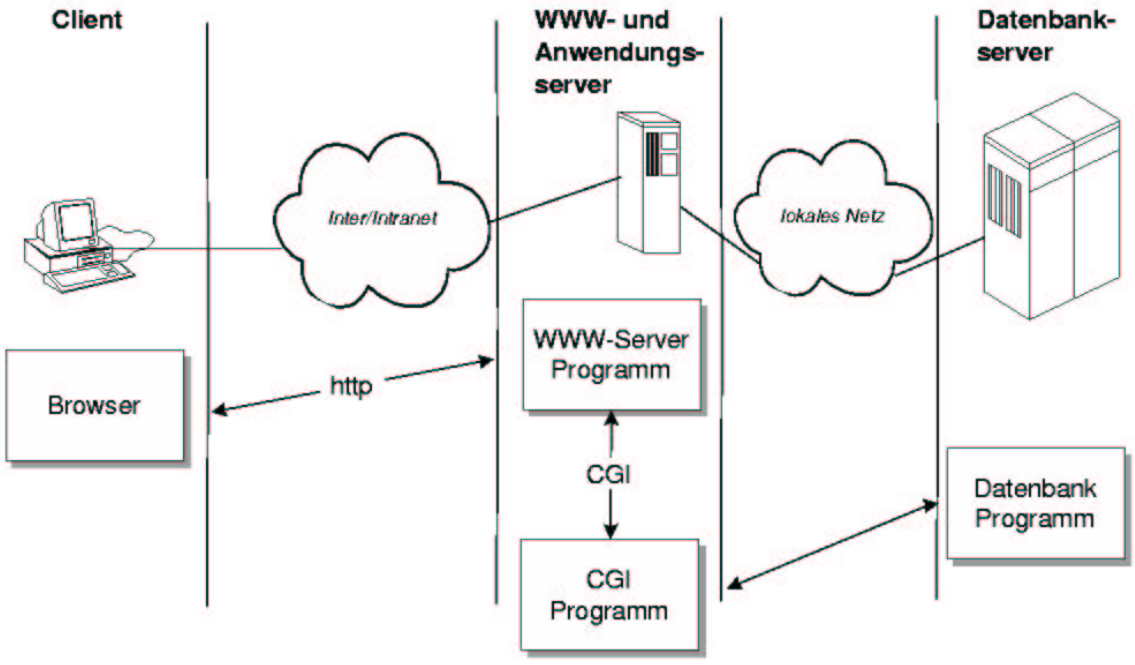


Abbildung 1.3: Der Ablauf bei der CGI-Schnittstelle

- *Java*: Java ist eine Programmiersprache, deren Laufzeitumgebung zur Zeit noch in alle marktgängigen Browser integriert ist. Sie ermöglicht es, spezielle in Java geschriebene Programme (Applets) über das Internet an den Client zu schicken und dort ausführen zu lassen. Da Java über ein ausgefeiltes Sicherheitskonzept verfügt, ist der Einsatz von Java auf dem Client unproblematisch.

- *Active X*: Active X beruht auf der Komponententechnologie von Microsoft. ActiveX Controls sind ausführbare Programme, die in die html-Seite eingebunden sind. Ein ausgefeiltes Sicherheitskonzept, wie in Java, existiert nicht. Bei falscher Einstellung können Eindringlinge volle Kontrolle über den Rechner erhalten.

- *JavaScript*: Durch JavaScript kann man im Prinzip auf alle Komponenten des Browsers und der dargestellten html-Seite zugreifen und im Nachhinein[3] Änderungen vornehmen. So könnte man z.B. in Abb. 1.2 durch JavaScript-Programme überprüfen lassen, ob in den Eingabefenstern überhaupt Eintragungen sind und wenn welche existieren, ob diese sinnvoll sind (Matrikelnummern sind bei uns beispielsweise 12-stellig und bestehen nur aus Zahlen). Darüber hinaus lassen sich wiederholende Abläufe automatisieren. Ein Beispiel hierfür: Normalerweise trägt jeder Internet-Autor das Datum der letzten Änderung eines Dokuments in seine Seiten ein. So etwas kann man natürlich auch einmal vergessen. Wird folgendes JavaScript-Programm in

[3]Dies hängt im Prinzip nur von der Browser-Version ab

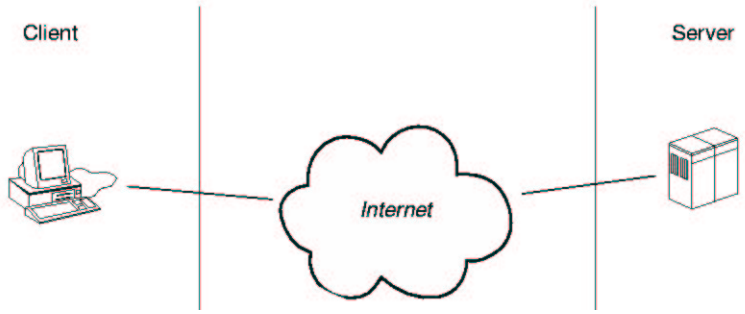


Abbildung 1.4: Der Client und der Server

die Seite eingefügt, erfolgen die Änderungen automatisch[4]:

```
<script language = "JavaScript">
     var dateChanged=new Date(document.lastmodified);
     var day = dateChanged.getDate();
     var month = dateChanged.getMonth());
     var year = dateChanged.getFullYear();
     document.write(day + "." + month + ". " + year);
</script>
```

Serverseitige Technologien werden entwickelt, da die CGI-Schnittstelle nicht wirklich einfach zu programmieren und Darüber hinaus auch langsam ist. Beispiele serverseitiger Technologien sind u.a.:

- *php*: Hierbei handelt es sich um eine speziell für Internet-Programmierung entwickelte Sprache.
- *jsp*: Dies sind Java-Programme, die in eine html-Seite eingefügt und auf dem Server ausgeführt werden. Hierfür muß eine Ausführungsumgebung auf dem Server bereitgestellt werden. Beispiele solcher Umgebungen sind: Tomcat, BEA-Weblogic, Websphere, Enhydra und JBoss.
- *asp*: asp ist eine Microsoft-Technologie zur Erzeugung dynamischer Webseiten. Die Programmierung erfolgt zumeist in Visual Basic Script.

Ausführungsumgebungen für die oben dargestellten Technologien werden auch als Application-Server bezeichnet.

[4]Dies Programm brauchen Sie jetzt noch nicht zu verstehen, dies lernen Sie später

Kapitel 2

Erste Beispiele

In diesem Kapitel stelle ich einige kleinere Programme vor. Dies soll Ihnen ein „Gefühl“ für die Entwicklung von Webanwendungen geben. Sie sollen die Beispiele nicht vollständig verstehen, alle Programmkonstrukte, die in diesem Kapitel vorkommen, werden in eigenen Kapiteln eingehend behandelt. Was Sie jedoch sehr wohl verstehen sollten, ist der Unterschied zwischen client- und serverseitiger Entwicklung.

Wir beginnen (wie in Büchern über Programmierung allgemein üblich) mit einem Programm, welches „Hello World“ ausgibt.

Beispiel 2.1 *Das Programm „Hello World“ in JavaScript*

```
<!-- Das Programm gibt "Hello World" im Browser aus
  Dateiname: helloWorld.html //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Hello World</title>
</head>
<body>
    <script LANGUAGE = "JavaScript">
        document.write ("<H2> Hello World </H2>");
    </script>
</body>
</html>
```

Dies Programm erzeugt im Browser das in Abb. 2.1 dargestellte, wenig überraschende Ergebnis:

Beispiel 2.1 ist sicher nicht das sinnvollste Programm, denn eine Browserausgabe, wie in 2.1 dargestellt, können wir sicherlich einfacher erhalten. Eigentlich ist der in Beispiel 2.2 dargestellte html-Code völlig ausreichend:

Beispiel 2.2 *„Hello World“ in html*

```
<!-- Das Programm gibt "Hello World" im Browser aus
  Dateiname: helloWorld2.html //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
```

Abbildung 2.1: Ausgabe von Beispiel 2.1

```
<head>
  <title>Hello World</title>
</head>
<body>
    <H2> Hello World </H2>
</body>
</html>
```

Dennoch wollen wir Beispiel 2.1 kurz durchgehen. Die ersten acht Zeilen sind Standard-html. In der neunten Zeile beginnt JavaScript. Das html-Tag `<script language= ''JavaScript''>` sagt dem die html-Datei darstellenden Browser, dass nun JavaScript-Code folgt.`document.write` ist ein JavaScript Befehl[1], der html-Text an den Browser übergibt und den Browser veranlasst, diesen html-Text darzustellen. Der Browser stellt also jetzt `<H2> Hello World </H2>` dar und es erscheint Hello World in der Formatierungsvorschrift H2 im Browser.

Nun wollen wir die gleiche Problematik in php lösen:

Beispiel 2.3 *Das Programm „Hello World“ in php*

```
<!-- Das Programm gibt "Hello World" im Browser aus
  Dateiname: helloWorld.php //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Hello World</title>
</head>
<body>
<?php
        echo "<h2> Hello World </h2>";
?>
</body>
</html>
```

Zunächst fällt auf – wie man dem Kommentar entnehmen kann –, dass die Extension unserer Beispieldatei nicht mehr html, sondern php ist. Dies liegt daran, dass php eine vom Server ausgeführte

[1] Was streng genommen gar nicht richtig ist, da document.write ein Funktions- oder Methodenaufruf ist. Doch was das ist, erfahren Sie erst sehr viel später.

Sprache ist. An der Extension php erkennt die Web-Server-Software, dass diese Datei nicht so, wie sie ist, über das Internet an den Client geschickt werden soll. Vielmehr analysiert die Web-Server-Software zunächst unsere Datei.

Jeweils zwischen den Tags `<?php` und `?>` erwartet die Web-Server-Software gültigen php-Code. Dieser Code wird von der Web-Server-Software an das php-Modul weitergeleitet und dort ausgeführt. Das Ergebnis des Programmlaufes, also das, was das php-Modul zurückgibt, wird anstelle des php-Codes in die analysierte Datei eingefügt. Das so entstandene Ergebnis schickt der Web-Server dann an den Browser. Machen wir uns dies an Beispiel 2.3 klar. Hier sehen wir in Zeile 9 den einleitenden `<?php`-Tag. In Zeile 10 folgt nun die einzige php-Anweisung des Programms:

```
echo "<h2> Hello World </h2>";
```

`echo` ist einer der leichteren php-Befehle. Alles was hinter `echo` steht wird einfach ausgegeben. Zeichenketten werden dabei, wie in JavaScript auch, in Anführungszeichen gesetzt. In der nächsten Zeile wird durch `?>` die Programmausführung beendet. Die Server-Software ersetzt also

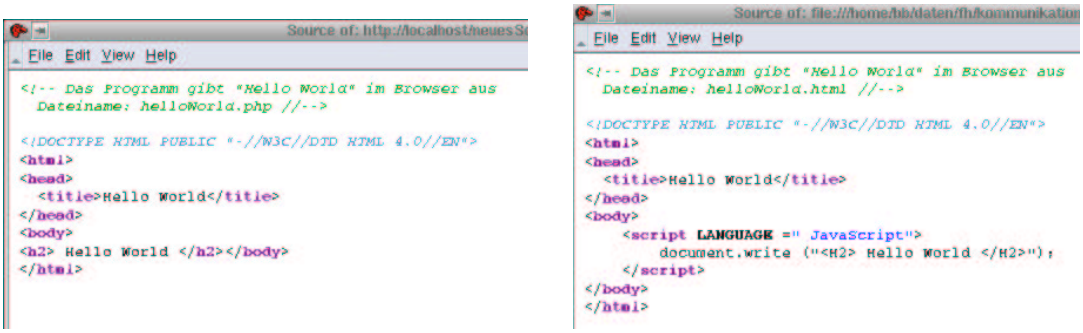
```
<?php
        echo "<h2> Hello World </h2>";
?>
```

durch

```
<h2> Hello World </h2>
```

und schickt die so entstandene Datei an den Client. Die so entstandene Datei ist also mit Beispiel 2.2 identisch und führt zu der in Abb. 2.1 dargestellten Ausgabe.

Um uns den Unterschied zwischen der php- und der JavaScript-Lösung noch einmal klar zu machen, schauen wir uns an, was beim Browser ankommt. Dazu müssen wir ja nur die „View Source“-Funktion des Browsers nutzen. Abb. 2.2 zeigt das Ergebnis.



Source dar php−Lösung

Source der JavaScript−Lösung

Abbildung 2.2: Die „View Source“ Funktion des Browsers angewendet auf unsere Lösungen

Bei der php-Lösung wird also das *Ergebnis eines Programmlaufs* auf dem Server an den Client übertragen. Der Browser „weiß“ gar nicht, dass er eine berechnete Seite darstellt. Clientseitig besteht kein Unterschied zu statischen html-Seiten.

Bei der JavaScript-Lösung hingegen wird das *Programm* selber an den Client übertragen. Der Browser führt das Programm aus und stellt das Ergebnis der Programmausführung dar. Der Browser muss also die Programmiersprache JavaScript ausführen können.

Betrachten wir nun ein weiteres Beispiel. Da Programmieren nach landläufiger Meinung häufig etwas mit Rechnen zu tun hat, schreiben wir nun ein kleines Rechenprogramm. Um uns nicht so ganz zu Anfang zu überfordern, beschränken wir uns darauf, ein Programm zu schreiben, das die Zahlen neun und zehn[2] addiert.

Schauen wir uns zunächst die JavaScript-Lösung an:

Beispiel 2.4 *Ein supereinfaches Additionsprogramm in JavaScript*

```
<!-- Das Programm addiert die zahlen 9 und 10
  Dateiname: addition1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition zum Ersten</title>
</head>
<body>
    <script language = "JavaScript">
        var ersterSummand;
        var zweiterSummand;
        ersterSummand=9;
        zweiterSummand=10;
        summe=ersterSummand+zweiterSummand;
        document.write ("Die Summe von " + ersterSummand +
                                " und " + zweiterSummand +
                                " ist: " + summe);
    </script>
</body>
</html>
```

Hier sehen wir neue Dinge. Wir haben 3 relativ gleich aussehende Zeilen:

```
var ersterSummand;
var zweiterSummand;
var summe;
```

Diese Zeilen beginnen mit dem Schlüsselwort `var`. `var` leitet eine Variablendeklaration ein. Variablen werden in Kapitel 4.2 ausführlich erklärt. Grob gesagt sind Variablen die programminternen Namen für die Dinge, mit denen das Programm arbeiten soll. Mit den Variablen kann dann beschrieben werden, was mit den Dingen beim Programmlauf gemacht werden soll.

Hier werden also drei Variablen deklariert mit den Namen ersterSummand, zweiterSummand und summe.

Danach folgen die Zuweisungen

```
ersterSummand=9;
zweiterSummand=10;
```

Dadurch werden den Variablen ersterSummand und zweiterSummand Werte zugewiesen. ersterSummand erhält den Wert 9, zweiterSummand den Wert 10. Durch die Zeile

```
summe=ersterSummand+zweiterSummand;
```

[2] Das Ergebnis ist übrigens 19!

wird die Summe der Werte der Variablen ersterSummand und zweiterSummand der Variablen summe zugewiesen. Da ersterSummand den Wert 9 hatte und zweiterSummand den Wert 10, wird auf der Variablen summe nun der Wert 19 abgespeichert.

Die Zeile

```
document.write ("Die Summe von " + ersterSummand +
                                " und " + zweiterSummand +
                                " ist: " + summe);
```

kennen wir im Wesentlichen bereits. Sie dient zur Ausgabe des Ergebnisses. Allerdings sehen wir auch hier Neues. Dies deswegen, weil wir nicht nur das Ergebnis ausgeben wollen, sondern einen schönen Satz, der dem Betrachter sagt, was das Programm gerechnet hat. Wir müssen hier also zwei unterschiedliche Dinge auf den Bildschirm bringen:

- *Feststehende Zeichenketten:* “Die Summe von”, “ und ”, “ ist: ”
- *Die Werte oder Inhalte von Variablen:* ersterSummand, zweiterSummand und summe.

Zeichenketten werden grundsätzlich in Anführungszeichen gesetzt. Variablen, deren Werte ausgegeben werden sollen, dürfen nicht in Anführungszeichen stehen. Verbunden werden sie mit dem +-Zeichen.

Zudem können innerhalb eines JavaScript-Befehls[3] beliebig Zeilenschaltungen oder Einrückungen vorkommen, ohne die Ausgabe zu verändern[4].

Beispiel 2.4 liefert natürlich das in Abb 2.3 dargestellte Ergebnis.

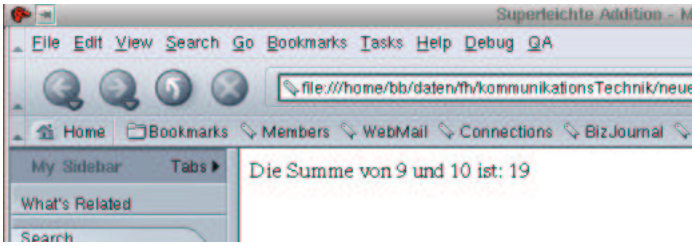


Abbildung 2.3: Ausgabe von Beispiel 2.4

Schauen wir uns nun die php-Lösung an:

Beispiel 2.5 *Ein supereinfaches Additionsprogramm in php*

```
<!-- Das Programm addiert die Zahlen 9 und 10
  Dateiname: addition1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Superleichte Addition</title>
</head>
<body>
<?php
    $ersterSummand=9;
```

[3] Wie schon gesagt hier richtiger eines Methodenaufrufs

[4] Ausnahme: Innerhalb von Anführungszeichen sind keine Zeilenschaltungen erlaubt.

```
    $zweiterSummand=10;
    $summe=$ersterSummand+$zweiterSummand;
    echo ("Die Summe von $ersterSummand und $zweiterSummand" .
                    " ist: $summe");
?>
</body>
</html>
```

Zunächst einmal überraschend ist, dass wir in php keine Variablen deklarieren. In php gibt es dazu keine Möglichkeit. Immer, wenn wir einen Variablennamen in die php-Quelle schreiben, erzeugt php diese Variable. Natürlich nur, wenn es die Variable nicht schon gibt. Variablen in php beginnen mit dem $-Zeichen.

Die Zeilen

```
$ersterSummand=9;
$zweiterSummand=10;
```

erzeugen die Variablen $ersterSummand, $zweiterSummand und weisen ihnen die Werte 9 resp. 10 zu.

```
$summe=$ersterSummand+$zweiterSummand;
```

Hier wird die Variable $summe neu erzeugt. Da die Variablen $ersterSummand und $zweiterSummand bereits existieren, versucht php nicht, diese Variablen neu zu erstellen. Anstelle dessen werden die Werte der bereits existierenden Variablen genommen und addiert. Das Ergebnis (19) wird der neuen Variable $summe zugewiesen.

Zum Schluss erfolgt die Ausgabe:

```
echo ("Die Summe von $ersterSummand und $zweiterSummand" .
                " ist: $summe");
```

Den echo-Befehl kennen Sie schon aus Beispiel 2.3. Im Gegensatz zu JavaScript können wir hier Variablen und normalen Text in den Anführungszeichen mischen. Der Grund ist der Zwang, Variablennamen mit dem $-Zeichen beginnen zu lassen. Hieran erkennt php, dass wir keinen normalen Text mehr meinen, sondern dass nun der Wert einer Variable ausgegeben werden soll[5]. Auch in php ist der Ausgabetext zu lang für eine Zeile und wie in JavaScript können wir beliebig Zeilenschaltungen einbauen, nur nicht innerhalb von Anführungszeichen. Wir zerlegen also unsere Zeichenkette in zwei Zeichenketten. Der Operator[6], um zwei Zeichenketten miteinander zu verbinden, ist in php der Punkt (.). Zur Erinnerung: In JavaScript war es das +-Zeichen.

Sie sehen, dass der Ausgabetext in Beispiel 2.5 im Gegensatz zu Beispiel 2.3 in runden Klammern eingeschlossen ist. Normalerweise können runde Klammern beim echo-Befehl weggelassen werden. Ausnahme ist, wenn sich der der echo-Befehl über mehrere Zeilen erstreckt (wie in Beispiel 2.5) [7].

Wie im vorherigen Beispiel sehen wir uns auch hier die Quellen der Seiten, wie sie der Browser sieht, an.

Abb. 2.4 zeigt uns den selben Sachverhalt wie Abb. 2.2. Bei der Verwendung von JavaScript wird wieder das von uns geschriebene Programm an den Browser geschickt, der Browser führt es aus[8], das Ergebnis der Programmausführung des Browsers wird dem Benutzer dargeboten.

[5] Wie Sie das $-Zeichen selbst ausgeben, lernen Sie später

[6] Auch was ein Operator genau ist, werden Sie später lernen

[7] Dies sind die Feinheiten, die programmieren so spannend machen :-).

[8] muss daher JavaScript beherschen

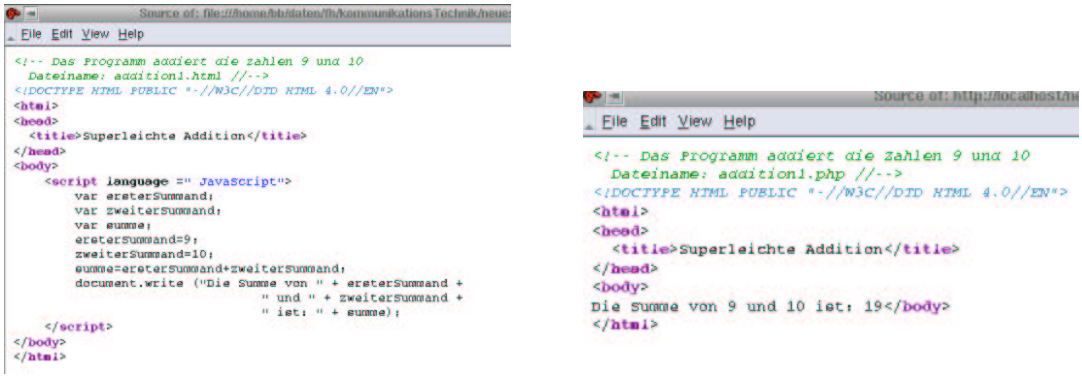


Abbildung 2.4: Die „View Source“ Funktion des Browsers angewendet auf unsere Lösungen

Bei php ist dies grundsätzlich anders. Der Programmcode wird von einem Modul der Serversoftware ausgeführt. Das Ergebnis wird in die php-Datei zurückgeschrieben und an den Browser übergeben. Der Browser erhält also reines html und kann dies ohne eigene Programmausführung darstellen.

Natürlich ist auch dieses Beispiel noch nicht so hochintelligent, denn mit der statischen html-Datei aus Beispiel 2.6 hätten wir dasselbe Ergebnis erhalten.

Beispiel 2.6 *Das supereinfache Additionsprogramm in html*

```
<!-- Das Programm addiert die Zahlen 9 und 10
  Dateiname: addition1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Superleichte Addition</title>
</head>
<body>
    <p>
        Die Summe von 9 und 10 ist: 19
    </p>
</body>
</html>
```

Doch kommen wir nun zu einem, in statischem html nicht mehr abbildbaren, Beispiel. Wir verallgemeinern 2.4. Unser Programm soll nicht auf die Zahlen 9 und 10 beschränkt bleiben, sondern beliebige Zahlen addieren. Die Benutzer sollen über ihren Browser die Zahlen eingeben, unser Programm soll die Summe der eingegebenen Zahlen ausrechnen und das Ergebnis ausgeben. Es leuchtet sofort ein, dass eine solche Aufgabe mit „normalem“ html nicht mehr lösbar ist. Denn der Autor der Web-Seiten kann ja nun nicht wissen, welche Zahlen gerade addiert werden sollen. Und html-Seiten für alle möglichen Zahlenkombinationen vorzuhalten ist ziemlich unmöglich, da es, wie wir alle ja aus der Mathematik wissen, unendlich[9] viele Zahlen gibt.

Was wir hier also benötigen, ist eine Interaktionsmöglichkeit des Programms mit dem Benutzer.

[9] ja sogar unabzählbar

In JavaScript ist das relativ einfach. JavaScript wird durch den Browser des Benutzers ausgeführt und verfügt über zahlreiche Funktionen zur Interaktion mit den Benutzern. Eine Lösung ist in Beispiel 2.7 dargestellt.

Beispiel 2.7 *Addition zweier einzugebender Zahlen*

```
<!-- Das Programm addiert zwei einzugebende Zahlen
  Dateiname: addition2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe </title>
</head>
<body>
    <script language = "JavaScript">
        var ersterSummand;
        var zweiterSummand;
        var summe;
        ersterSummand=prompt("Bitte geben Sie den ersten Summanden ein!","");
        zweiterSummand=prompt("Bitte geben Sie den zweiten Summanden ein!","");
        ersterSummand=parseFloat(ersterSummand);
        zweiterSummand=parseFloat(zweiterSummand);
        summe=ersterSummand+zweiterSummand;
        document.write ("Die Summe von " + ersterSummand +
                                " und " + zweiterSummand +
                                " ist: " + summe);
    </script>
</body>
</html>
```

Der größte Teil des Programmcodes ist uns schon bekannt. Der erste Unterschied sind die Zeilen:

```
ersterSummand=
  prompt("Bitte geben Sie den ersten Summanden ein!","");
zweiterSummand=
  prompt("Bitte geben Sie den zweiten Summanden ein!","");
```

Die JavaScript-Funktion `prompt` blendet ein Eingabefenster auf. Das Fenster besteht aus einem Textteil und einem Eingabefeld (vgl. Abb. 2.5).

Der erste Parameter der Funktion `prompt` (in unserem Fall “Bitte geben Sie den ersten Summanden ein!”) wird im Textteil dargestellt. Der zweite Parameter (in unserem Fall “”, also leer) ist der Vorgabewert für das Eingabefeld.

Der Inhalt dieses Eingabefeldes wird, wenn der Benutzer den OK-Button clickt, der Variable ersterSummand zugewiesen. Die Zeilen

```
ersterSummand=
  prompt("Bitte geben Sie den ersten Summanden ein!","");
zweiterSummand=
  prompt("Bitte geben Sie den zweiten Summanden ein!","");
```

sind übrigens Zuweisungen (wird in Kapitel 4.4.1 behandelt). Nachdem beide Eingabefenster wie in Abb. 2.5 ausgefüllt und mit clicken des OK-Buttons abgeschickt worden sind, ist der Wert der Variablen ersterSummand 5 und der Wert der Variablen zweiterSummand 9.

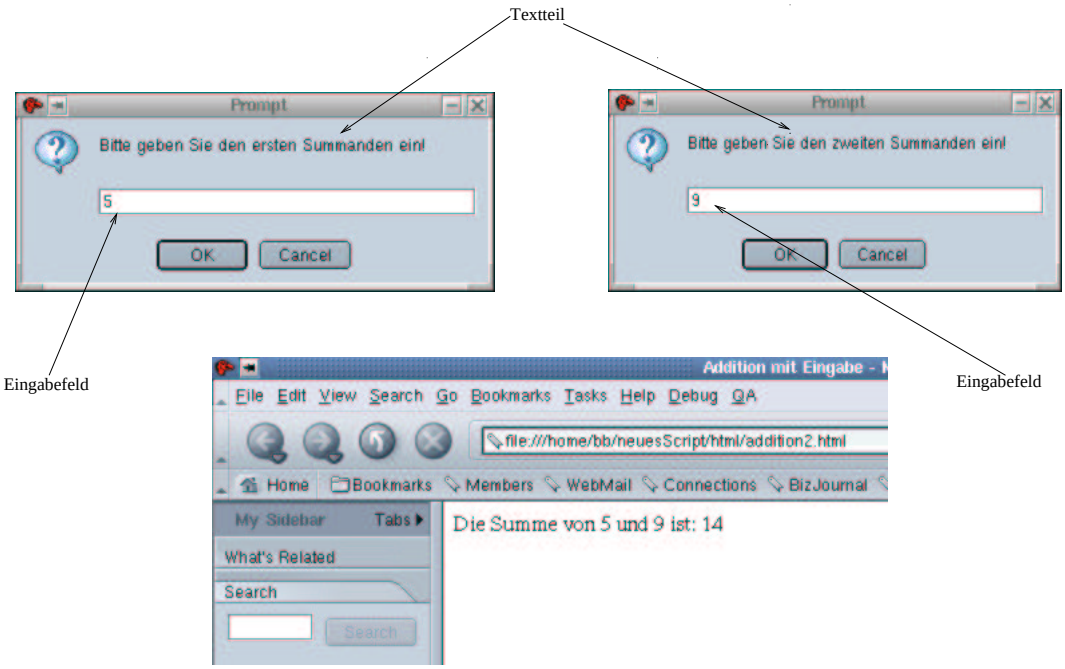


Abbildung 2.5: Ein- und Ausgaben von Beispiel 2.5

Allerdings haben wir noch ein Problem: `prompt` gibt die Eingabe des Benutzers als Zeichenkette (String) zurück. Und mit Strings kann man nicht rechnen. Also müssen die Zeichenketten noch in Zahlen umgewandelt werden. Dafür gibt es die JavaScript-Funktion `parseFloat`. Die Zeilen

```
ersterSummand=parseFloat(ersterSummand);
zweiterSummand=parseFloat(zweiterSummand);
```

wandeln die eingelesenen Strings in Zahlen um. Wir erkennen hier weitere wichtige Prinzipien der Programmierung:

- Eine Variable kann auf der rechten und linken Seite einer Zuweisung stehen. Es wird immer zuerst die rechte Seite ausgewertet. Das Ergebnis der Auswertung wird der Variablen auf der linken Seite zugewiesen.
- Variablen können im Programmverlauf ihren Wert ändern.
- In JavaScript können Variablen sogar ihren Typ ändern. Die Variable ersterSummand war zunächst ein String, nach der Umwandlung durch `parseFloat` ist es eine Fließkommazahl.

Wie im vorherigen Beispiel sehen wir uns auch hier die Quelle der JavaScript-Seite, wie sie an den Browser übermittelt wird, an.

Abb. 2.6 zeigt uns den selben Sachverhalt wie in den vorherigen Beispielen. Bei der Verwendung von JavaScript wird wieder das von uns geschriebene Programm an den Browser geschickt, der Browser führt es aus[10], das Ergebnis der Programmausführung des Browsers wird dem Benutzer dargeboten.

Kommen wir nun zur Realisierung in php. Hier stellen sich die Dinge ungleich komplizierter dar. php wird ja, wie ich nun schon öfter erwähnt habe, auf dem Server ausgeführt. Das bedeutet, unser

[10] muss daher JavaScript beherschen

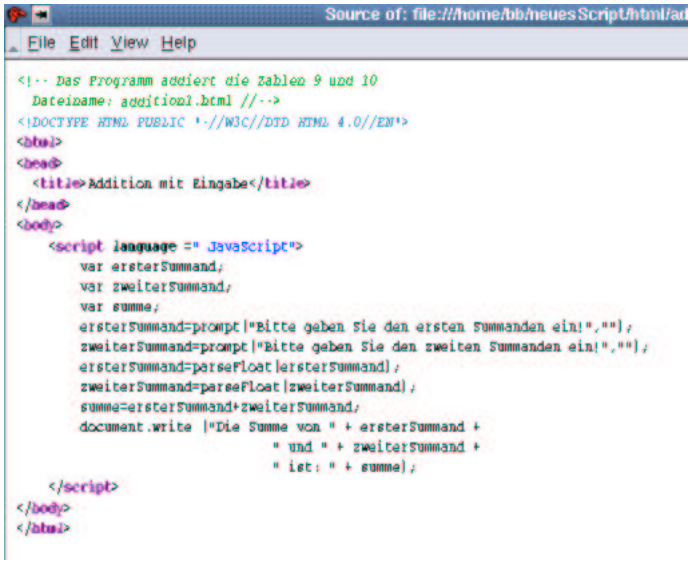


Abbildung 2.6: Die „View Source“ Funktion des Browsers angewendet auf die JavaScript-Lösung

Programm muss bei der Ausführung eigentlich schon wissen, welche Zahlen der Benutzer eingegeben hat. Eingabefenster, wie sie bei JavaScript möglich sind, gibt es in php nicht.

Die Lösung erfolgt hier durch html-Formulare, html-Eingabefenster und den Aufruf einer neuen Seite durch das `action`-Attribut des `form`-Tags. Doch sehen wir uns die Lösung im Einzelnen an. Wir erstellen zunächst eine html-Seite, die ein Formular enthält, in das wir die beiden Summanden eingeben können. Beispiel 2.8 zeigt dies. Beispiel 2.8 erzeugt die in Abb. 2.7 dargestellte Browser-Seite.

Beispiel 2.8 *Addition zweier einzugebender Zahlen mit php (Teil 1)*

```
<!-- Das Programm addiert zwei einzugebende Zahlen
  Dateiname: addition2php.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe in php</title>
</head>
<body>
<h2>
    Addition zweier Zahlen
</h2>
<form name="addition" action="./addition2php.php" method="POST">
    <table border>
        <tr>
            <td>
                Erster Summand
            </td>
            <td>
                <input type="text" name="ersterSummand" size=12>
            </td>
        </tr>
        <tr>
```

```
            <td>
                Zweiter Summand
            </td>
            <td>
                <input type="text" name="zweiterSummand" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Abbildung 2.7: Die Browser-Darstellung von Beispiel 2.8

Wie Sie bereits gelernt haben, führt das Clicken auf den Submit-Button dazu, dass der Inhalt der Eingabefelder an den Server übertragen wird, der Server das im `action`-Attribut des `form`-Tags angegebene Programm startet und diesem Programm die Inhalte der Eingabefelder übergibt. Neu ist, dass das vom Server gestartete Programm eine php-Datei sein kann. Bleibt also noch, diese php-Datei zu implementieren.

Beispiel 2.9 *Addition zweier einzugebender Zahlen mit php (Teil 2)*

```
<!-- Das Programm addiert zwei einzugebende Zahlen
  Dateiname: addition2php.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe in php (Teil 2)</title>
</head>
<body>
<h2>
    Addition zweier Zahlen: Das Ergebnis
</h2>
```

```
<?php
    $summe=$ersterSummand+$zweiterSummand;
    echo ("Die Summe von $ersterSummand und $zweiterSummand" .
                    " ist: $summe");
?>
</body>
</html>
```

Beispiel 2.9 erzeugt das in Abb. 2.8 dargestellte Ergebnis.

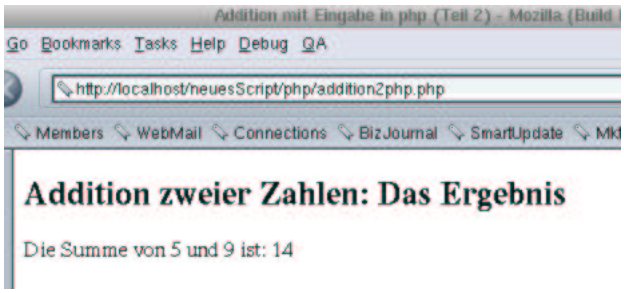


Abbildung 2.8: Die Browser-Darstellung von Beispiel 2.8

Dies Ergebnis riecht verdächtig nach schwarzer Magie. Woher wusste das in Beispiel 2.9 dargestellte php-Programm, welche Zahlen ich in die Eingabefelder eingegeben habe? Die Antwort ist dann aber an sich doch leicht verständlich. Schauen wir uns dazu die Definition der beiden Eingabefelder in Beispiel 2.8 an:

```
<input type="text" name="ersterSummand" size=12>
<input type="text" name="zweiterSummand" size=12>
```

Betrachten wir die `name`-Attribute der `input`-Tags. Hier werden den Eingabefeldern Namen gegeben. Diese Namen stimmen seltsamerweise fast mit den Variablennamen in Beispiel 2.9 überein. Einziger Unterschied ist, dass den Namen hier ein $-Zeichen vorangestellt ist. Das muss aber so sein, denn wie Sie ja bereits wissen, beginnen alle Variablennamen in php mit dem $-Zeichen. Des Rätsels Lösung ist nun, dass bei der Übertragung des Formulars das php-Modul des Servers automatisch Variablen mit dem Namen der Eingabefelder anlegt. Diese Variablen erhalten den vom Benutzer eingegebenen Wert zugewiesen und wir können sie in unserem php-Code benutzen.

Der Rest des php-Codes entspricht Beispiel 2.5. Auffällig ist noch, dass wir hier nicht wie in JavaScript den Typ der Variablen ändern müssen. php erkennt, dass es sich um Zahlen handelt und behandelt unsere Variablen automatisch richtig.[11]

Um Interaktivität in php zu realisieren, müssen Sie also zwei Dateien[12] programmieren. Eine Seite, in der die Benutzer ihre Eingaben machen können, und eine Seite, die die Eingaben verarbeitet.

Kommem wir nun zu einem weiteren Beispiel: Wir möchten alle natürlichen Zahlen bis zu einer einzugebenden Zahl aufsummieren. Schauen wir uns zunächst die JavaScript-Realisierung an:

Beispiel 2.10 *Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl mit JavaScript*

[11] Der wahre Grund, warum wir in JavaScript umwandeln müssen, liegt in der Doppelbedeutung des +-Zeichens in JavaScript – dies wird in Kapitel 4.4 erklärt –. Hätten wir multipliziert, hätte JavaScript die Umwandlung ebenfalls automatisch vorgenommen. Sie können ja mal die Typwandlung in Beispiel 2.7 weglassen und sich das Ergebnis im Browser anschauen.

[12] Dies ist strenggenommen nicht ganz richtig. Sie werden in Kapitel 6.2.3 erfahren, wie es auch mit einer Seite geht.

```
<!-- Das Programm addiert natuerliche Zahlen
    bis zu einer eingegebenen Zahl
  Dateiname: addition3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe</title>
</head>
<body>
    <script language = "JavaScript">
        var bisZu;
        var summe;
        bisZu=prompt("Bitte geben Sie Zahl, bis zu der summiert wird, ein","");
        bisZu=parseInt(bisZu);
        summe=0;
        for(i=1;i<=bisZu;i++)
        {
            summe=summe+i;
        }
        document.write ("Die Summe der ersten " + bisZu +
                              " nat&uuml;rlichen Zahlen " +
                               " ist: " + summe);
    </script>
</body>
</html>
```

Dieses Programm erzeugt ein Eingabefenster, in das wir die Zahl bis zu der addiert werden soll eingeben können, und stellt daraufhin das berechnete Ergebnis im Browser dar. Abb. 2.9 zeigt eine beispielhafte Ein- und Ausgabe.

Abbildung 2.9: Beispielhafte Ein-und Ausgabe von Beispiel 2.10

Die ersten Zeilen sind Ihnen schon vertraut (zumindestens haben Sie ähnliches jetzt bereits einige Male gesehen). Zunächst werden zwei Variablen deklariert, danach wird die Variable eingelesen, bis zu der aufsummiert werden soll. In der nächsten Zeile wird der Variablentyp gewechselt. Wir benutzen hier die JavaScript-Funktion `parseInt`. `parseInt` wandelt in eine ganze Zahl um, `parseFloat` in eine Fließkommazahl[13]

Dann jedoch kommt Neues. Zunächst wir die Variable summe mit 0 initialisiert.

```
summe = 0
```

Dann folgt eine Schleife. Schleifen werden in Kapitel ?? ausführlich erklärt. Schleifen sind Teile

[13]Der Unterschied sollte Ihnen aus der Mathematik noch geläufig sein.

unseres Programmcodes, die mehrfach durchlaufen (dies bedeutet durchgeführt) werden können. Die Schleife beginnt mit dem Befehl[14]:

```
for(i=1;i<=bisZu;i++)
```

Trifft JavaScript auf diese Zeile, wird die Variable i mit dem Wert eins initialisiert. Danach wird überprüft, ob der Wert der Variablen i kleiner gleich dem Wert der Variablen bisZu ist. Wenn der Benutzer wie in Abb. 2.9 vier eingegeben hat, ist dies der Fall. Daher werden nun die Anweisungen der Schleife abgearbeitet. Dies sind alle Anweisungen, die sich zwischen den öffnenden und schließenden geschweiften Klammern befinden. In unserem Beispiel ist dies nur die Anweisung:

```
summe=summe+i;
```

Da die Variable summe mit dem Wert Null und i mit dem Wert eins initialisiert wurde, ergibt summe + i (0 + 1) den Wert eins. Dieser neue Wert wird der Variablen summe zugewiesen. summe hat also jetzt den Wert eins. Dann wird das Ende der Schleife (die schließende geschweifte Klammer) erreicht. Hierdurch wird zunächst der Wert der Variablen i um eins erhöht. Da i vorher den Wert eins hatte und eins plus eins zwei ergibt, hat i danach den Wert zwei. Des Weiteren veranlasst die schließende geschweifte Klammer JavaScript zu der Anweisung

```
for(i=1;i<=bisZu;i++)
```

zurückzugehen. Da diese Anweisung nun zum zweiten Mal durchgeführt wird, wird die Initialisierung von i nicht mehr durchgeführt. Dies passiert nur bei der ersten Durchführung der `for`-Anweisung. i behält also den Wert 2. for überprüft nur , ob der Wert von i immer noch kleiner oder gleich dem Wert von bisZu ist. Da i 2 ist und bisZu 4, ist dies der Fall. Die Anweisung in der Schleife wird durchgeführt. Die Anweisung der Schleife war:

```
summe=summe+i;
```

Da summe nach dem letzten Schleifendurchlauf den Wert eins zugewiesen erhielt und i zur Zeit den Wert zwei hat ergibt summe + i (1 + 2) nun drei. Dieser neue Wert wird der Variablen summe zugewiesen. Durch die schließende geschweifte Klammer wird i um eins erhöht (i ist jetzt drei) und JavaScript kehrt zur for-Anweisung zurück. Hier wird erneut überprüft, ob der Wert von i immer noch kleiner oder gleich dem Wert von bisZu ist. Dies ist der Fall (i ist drei, bisZu immer noch vier), also wird wieder die Anweisung in der Schleife durchgeführt. summe war nach dem letzten Schleifendurchlauf 3, i ist zur Zeit auch 3, also ergibt summe + i (3 + 3) 6. Dieser neue Wert wird der Variablen summe zugewiesen.

Die schließende geschweifte Klammer erhöht i wieder um eins, (i ist jetzt vier) und JavaScript kehrt zur `for`-Anweisung zurück. Hier wird erneut überprüft, ob ob der Wert von i immer noch kleiner oder gleich dem Wert von bisZu ist. Dies ist der Fall (i ist vier, bisZu immer noch vier), also wird wieder die Anweisung in der Schleife durchgeführt. summe war nach dem letzten Schleifendurchlauf sechs, i ist zur Zeit vier, also ergibt summe + i (6 + 4) zehn. Dieser neue Wert wird der Variablen summe zugewiesen.

Die schließende geschweifte Klammer erhöht i wieder um 1, (i ist jetzt fünf) und JavaScript kehrt zur for-Anweisung zurück. Hier wird erneut überprüft, ob ob der Wert von i immer noch kleiner oder gleich dem Wert von bisZu ist. Dies ist diesmal nicht der Fall (i ist fünf, bisZu immer noch vier). Dies veranlasst JavaScript nun, die Schleife zu beenden. Eine Schleife zu beenden bedeutet, mit der ersten Anweisung hinter der Schleife fortzufahren. Dies ist:

[14] Die Anweisungen eines Programms werden auch Befehle genannt

Tabelle 2.1: Werte der Variablen i und summe während des Programmlaufs (Wertetabelle)

Schleifendurchlauf	Wert der Variablen i	Wert der Variablen summe
1	1	1
2	2	3
3	3	6
4	4	10

```
document.write ("Die Summe der ersten " + bisZu +
                            " nat&uuml;rlichen Zahlen " +
                             " ist: " + summe);
```

Unser berechnetes Ergebnis wird jetzt ausgegeben. summe hatte zum Schluss den Wert 10, also erscheint dieser auch im Browserfenster (vgl. Abb. 2.9).

Wir stellen also noch einmal fest:

- Die Werte von Variablen können sich ändern. Tabelle 2.1 zeigt noch einmal die Änderungen der Werte der Variablen summe und i während der Schleifendurchläufe.
- Variablen können auf der rechten und auf der linken Seite einer Zuweisung stehen.
- Es wird zuerst die rechte Seite einer Zuweisung ausgewertet. Das Ergebnis dieser Auswertung wird der Variablen auf der linken Seite zugewiesen.

Darüberhinaus haben wir einige neue Dinge kennengelernt:

- Programmteile können mehrfach durchlaufen werden.
- Variablen, die zuerst auf der rechten Seite einer Zuweisung vorkommen müssen initialisiert werden. Gäbe es nicht die Zeile:

  ```
  summe = 0
  ```

  könnte JavaScript nicht wissen, welchen Wert summe beim ersten Durchlaufen der Anweisung

  ```
  summe = summe + i
  ```

  besitzt.
- Schleifenvariablen (i ist die Schleifenvariable) müssen initialisiert werden.

Schauen wir uns nun die Realisierung dieses Beispiels in php an. Hier müssen wir, wegen der Probleme mit der Benutzerinteraktion bei serverbasierten Technologien, mit der zwei Dateien Methode arbeiten. Zunächst die html-Seite, in der wir die Eingaben machen können:

Beispiel 2.11 *Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl mit php*

```
<!-- Das Programm summiert Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: addition3php.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
```

```
<head>
  <title>Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl in ph
</head>
<body>
<h2>
    Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl
</h2>
<form name="addition3" action="./addition3php.php" method="POST">
    <table border>
        <tr>
            <td>
                Zahl, bis zu der summiert werden soll
            </td>
            <td>
                <input type="text" name="bisZu" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Diese Seite ist nicht weiter überraschend. Sie entspricht im wesentlichen Beispiel 2.8. Als nächstes müssen wir die Seite schreiben, die aufgerufen wird, wenn der Benutzer auf „Abschicken“ clickt. Der Name der Seite ist durch das `action`-Attribut des `form`-Tags festgelegt. Sie heisst „addition3php.php“.

Beispiel 2.12 *Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl mit php (Teil 2)*

```
<!-- Das Programm summiert Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: addition3php.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe in php (Teil 2)</title>
</head>
<body>
<h2>
    Aufsummierung nat&uuml;rlicher Zahlen: Das Ergebnis
</h2>
<?php
        $summe=0;
        for($i=1;$i<=$bisZu;$i++)
        {
            $summe=$summe+$i;
        }
```

```
        echo ("Die Summe der ersten $bisZu " .
                                " nat&uuml;rlichen Zahlen " .
                                 " ist: $summe");
?>
</body>
</html>
```

Die Realisierung unterscheidet sich nicht wesentlich von der Realisierung in JavaScript. Die Variablen beginnen alle mit dem $-Zeichen. Auf das Einlesen kann verzichtet werden, da über den Namen des Eingabefeldes (bisZu) eine php-Variable gleichen Namens erzeugt wurde, die den vom Benutzer eingegebenen Wert enthält.

Abb. 2.10 zeigt Ein- und Ausgaben der php-Lösung.

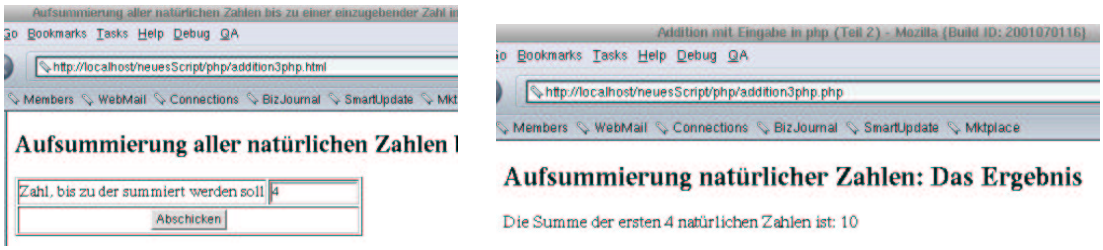


Abbildung 2.10: Beispielhafte Ein- und Ausgabe von Beispiel 2.11 und 2.12

Übrigens, wer nicht so gut in Programmierung, aber dafür gut in Mathematik ist [15], hätte Beispiel 2.4 sehr viel einfacher lösen können. Es gilt nämlich:

$$\sum_{i=1}^{n} x_i = \frac{n(n+1)}{2}$$

Oder umgangssprachlich ausgedrückt: Die Summe der ersten n natürlichen Zahlen ist n mal (n+1) und das Ergebnis davon durch 2[16]. Wir hätten Beispiel 2.4 also auch folgendermaßen codieren können:

Beispiel 2.13 *Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl (mit Formel)*

```
<!-- Das Programm summiert Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: addition3Formel.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe</title>
</head>
<body>
    <script language = "JavaScript">
        var bisZu;
```

[15] was aber eher selten ist

[16] das kann man auch beweisen. Sie sind ja vom Grundstudium her sicher so gut in Mathematik, dass Sie den Beweis selber durchführen können :-).

```
        var summe;
        bisZu=prompt("Bitte geben Sie Zahl, bis zu der summiert wird, ein","");
        bisZu=parseInt(bisZu);
        summe=(bisZu*(bisZu+1))/2;
        document.write ("Die Summe der ersten " + bisZu +
                              " nat&uuml;rlichen Zahlen " +
                               " ist: " + summe);
    </script>
</body>
</html>
```

Oder in php:

Beispiel 2.14 *Aufsummierung aller natürlichen Zahlen bis zu einer einzugebenden Zahl in php (mit Formel)*

```
<!-- Das Programm summiert Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: addition3phpFormel.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe in php (Teil 2)</title>
</head>
<body>
<h2>
    Aufsummierung nat&uuml;rlicher Zahlen: Das Ergebnis
</h2>
<?php
        $summe=($bisZu*($bisZu+1))/2;
        echo ("Die Summe der ersten $bisZu " .
                              " nat&uuml;rlichen Zahlen " .
                               " ist: $summe");
?>
</body>
</html>
```

Das ist natürlich sehr viel einfacher und läuft auch schneller. Die Zeilen

```
summe=(bisZu*(bisZu+1))/2;
```

bzw.

```
$summe=($bisZu*($bisZu+1))/2;
```

zeigen uns, dass JavaScript und php nicht nur addieren, sondern auch multiplizieren und dividieren können. Darüber hinaus kann man Klammern setzen.

Noch zwei Bemerkungen:

- In allen Beispielen ist der Code eingerückt. Alle Codezeilen, die zu einem Programm gehören, sind um eine Tabulatorposition eingerückt. Die Programmzeilen, die zu Schleifen gehören, sind eine weitere Tabulatorposition eingerückt. Dies macht Programme leichter lesbar. Fehler werden schneller gefunden. Sie sollten sich diesen Programmierstil auch angewöhnen.

- Variablen haben sprechende Namen. Soll heißen: Der Name einer Variablen lässt Rückschlüsse auf ihren Inhalt zu. Unsere Variablen heißen ersterSummand, zweiterSummand und summe und nicht etwa a, b, c, d, was ja kürzer wäre. Programme mit „guten“ Variablennamen sind aber weitaus leichter lesbar und viel verständlicher. Einzige Ausnahme ist die Schleifenvariable i. Schleifenvariablen nennen wir immer i, j oder k. Das macht jeder so und daher weiß man, wenn eine Variable dieses Namens in einem Programm vorkommt, ist es eine Schleifenvariable und die Verständlichkeit bleibt gewahrt.

Aufgabe 2.1 *Schreiben Sie ein Programm, dass Ihren Namen im Browser ausgibt.*

Aufgabe 2.2 *Dies war ja schon eine recht einfache Aufgabe. Können Sie sich ein noch einfachereres Programm vorstellen? Wenn ja, schreiben Sie es :-).*

Aufgabe 2.3 *Das nächste Programm soll 2 Zahlen einlesen und die zweite von der ersten abziehen! Das Ergebnis soll im Browser ausgegeben werden!*

Aufgabe 2.4 *Dieses Programm soll die Summe der ersten n negativen Zahlen bilden. Die Zahl n wird eingelesen. Die Summe soll im Browser ausgegeben werden! Erstellen Sie eine Lösung mit einer Schleife und eine ohne.*

Aufgabe 2.5 *Wie Sie in diesem Kapitel gesehen haben, wird der JavaScript-Code an den Browser übertragen, der php-Code nicht. Dadurch können sich die Benutzer den JavaScript-Code im Browser ansehen, den php-Code nicht. Welche Konsequenzen hat das für Sie als Programmierer?*

Kapitel 3

Einfügen von JavaScript und php in html-Dateien

3.1 Einfügen von JavaScript in html-Dateien

3.1.1 <script> und </script>

Der einfachste Weg, JavaScript-Code in eine html-Datei einzufügen, besteht darin, JavaScript-Anweisungen zwischen den html-Tags `<script language=''JavaScript''>` und `</script>`. direkt in die html-Datei aufzunehmen. Dies ist auch die in allen Beispielen von Kapitel 2 gewählte Vorgehensweise.

Abkürzend kann hier `language=''JavaScript''` weggelassen werden. Der Sinn in der Angabe der Scriptsprache liegt darin, dass es neben JavaScript andere Scriptsprachen für Internet-Browser gibt. Durch die Angabe der Script- Sprache kann der Browser entscheiden, ob er das Script ausführen kann, oder die Anweisungen des Scriptes besser ignorieren sollte.

3.1.2 Das src-Attribut des <script>-Tags

Eine weitere Möglichkeit ist, anstelle des JavaScript-Codes einen Dateinamen innerhalb des `script`-Tags anzugeben. Der Browser lädt die referenzierte Datei. Er verhält sich dabei genauso, als ob die in der referenzierten Datei enthaltenen JavaScript-Anweisungen direkt in der html-Datei stehen würden. Ich veranschauliche dies an Beispiel 3.1.

Zunächst die html-Datei:

Beispiel 3.1 *Die html-Datei zu Beispiel 3.1*

```
<!-- Das Programm addiert natuerliche Zahlen
    bis zu einer eingegebenen Zahl
  Dateiname: addition4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe</title>
</head>
<body>
    <script language = "JavaScript"
                    src="./javascript/addition4.js">
    </script>
```

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

Die zugehörige JavaScript-Datei:

Beispiel 3.2 *Die JavaScript-Datei zu Beispiel 3.1*

```
// Dies Programm addiert Zahlen bis
// zu einer eingegebenen Zahl
var bisZu;
var summe;
bisZu=prompt("Bitte geben Sie Zahl, bis zu der summiert wird, ein","");
bisZu=parseInt(bisZu);
summe=0
for(i=1;i<=bisZu;i++)
{
    summe=summe+i;
}
document.write ("Die Summe der ersten " + bisZu +
                    " nat&uuml;rlichen Zahlen " +
                        " ist: " + summe);
```

Beispiel 3.1 zeigt, dass die JavaScript-Datei über das `src` (Source)-Attribut des script-Tags in die html-Datei geladen wird. Die JavaScript-Datei heißt addition4.js. Die Extension js ist notwendig, zumindestens für manche Browser. addition4.js steht im Verzeichnis javascript unterhalb des Verzeichnisses, in dem die html-Datei (addition4.html) beheimatet ist. Deshalb wird im src-Attribut der relative Pfad zu addition4.js angegeben. Pfad- und Dateinamen werden in Anführungszeichen eingeschlossen.

Der Code in addition4.js entspricht dem JavaScript-Code zwischen den <`script language =“JavaScript”`> und </`script`>-Tags in Beispiel 2.10. Beispiel 2.10 und Beispiel 3.1 erzeugen dasselbe Ergebnis.

Die Auslagerung der JavaScript-Anweisungen in eine eigene Datei hat viele Vorteile:

- Die html-Datei wird kleiner und besser überschaubar.
- Die Erstellung von html- und JavaScript-Datei kann mit unterschiedlichen Werkzeugen erfolgen.
- Von mehreren html-Seiten genutzter JavaScript-Code wird in einer Datei vorgehalten. Dadurch wird:
  - Die Wartung des Codes (und der html-Seiten) vereinfacht.
  - Festplattenspeicherbedarf vermindert.
  - Die Ladezeit insgesamt verkürzt, da eine JavaScript-Datei, die von mehreren html-Seiten genutzt wird, nur einmal geladen wird.

JavaScript-Code, der nur von einer html-Seite genutzt wird, in eine eigene Datei auszulagern, ist jedoch nicht immer sinnvoll. Die html-Seite wird zwar besser überschaubar, doch dafür muss eine weitere Datei geöffnet werden. Bei kleineren Dateien überwiegt dieser Nachteil und wir werden JavaScript und html in einer Datei belassen. Doch sobald die Seiten größer werden oder wir Programme schreiben, die mehrfach genutzt werden können, lagern wir den Code in eigenen Dateien aus.

3.2 Einfügen von php in html-Dateien

Um php in eine html-Datei einzubinden gibt es grundsätzlich vier Tags:
Der **SGML-Stil**:

```
<?
    // unser php-Code
?>
```

Der **XML-Stil**:

```
<?php
    // unser php-Code
?>
```

oder

```
<?PHP
    // unser php-Code
?>
```

Der **ASP-Stil**:

```
<%
    // unser php-Code
%>
```

ASP ist eine von Microsoft eingeführte Konkurenz-Technik von php. Wie php-Seiten die Extension php haben, so enden ASP-Seiten mit asp. asp setzt den Internet-Information-Server voraus. asp-Seiten werden wie php-Seiten auf dem Server ausgeführt. Das Ergebnis der Programmausführung wird, wie in php, an den Client übermittelt. ASP-Seiten werden normalerweise mit VBScript[1] erstellt. Der ASP-Stil wurde eingeführt, um ASP-Entwicklern den Umstieg auf php zu erleichtern.

Der ASP-Stil erfordert allerding eine Änderung der php-Konfigurationsdatei, er steht bei uns nicht zur Verfügung. Wenn man php-Seiten nicht auf eigenen Server veröffentlicht, sondern auf dem Server eines Internet-Providers, ist auch nicht sicher, ob dieser Stil zur Verfügung steht. Daher sollte man von seiner Verwendung Abstand nehmen.

Der **JavaScript-Stil**:

```
<script languge="php" runat="server">
    // unser php-Code
</script>
```

Der JavaScript-Stil ist dann sinnvoll, wenn Sie den html-code ihrer php-Seiten mit Software, wie Frontpage[2] von Microsoft bearbeiten wollen. Solche Software verändert ganz gerne mal eigenmächtig den html-Code ihrer Seiten. Insbesondere in html-Seiten eingefügter php-Code ist nicht sicher vor solcher Software. Mit dem JavaScript-Stil eingefügter Code wird jedoch weitgehend ignoriert.

Wir verwenden immer und ausschliesslich den XML-Stil.

Wie bei Javascript auch, können externe php-Dateien eingebunden werden. Dies erfolgt mit dem `require_once`-Befehl. Ich zeige dies an Beispiel 2.12. Zunächst die html-Datei:

[1] Andere Programmiersprachen sind theoretisch auch möglich.

[2] Da Sie das wegen der großen Klasse dieser Software sicher nicht wollen, trifft diese Bemerkung auf Sie nicht zu!

Beispiel 3.3 *Die html-Datei zu Beispiel 2.12*

```
<!-- Das Programm summiert bis zu einer
    eingegebenen Zahl
  Dateiname: addition4php.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe in php (Teil 2)</title>
</head>
<body>
<h2>
    Aufsummierung nat&uuml;rlicher Zahlen: Das Ergebnis
</h2>
<?php
    require("./includes/addition4.inc.php");
?>
</body>
</html>
```

Die Datei mit dem zugehörigen php-Code:

Beispiel 3.4 *Die php-Datei zu Beispiel 2.12*

```
<?php
    $summe=0;
    for($i=1;$i<=$bisZu;$i++)
    {
        $summe=$summe+$i;
    }
    echo ("Die Summe der ersten $bisZu " .
                        " nat&uuml;rlichen Zahlen " .
                            " ist: $summe");
?>
```

Wir erkennen, dass in der eingefügten php-Datei ebenfalls die php-Tags stehen müssen. Die php-Datei heißt addition4.inc.php. Sie steht im Verzeichnis includes unterhalb des Verzeichnisses, in dem die html-Datei (addition4php.php) beheimatet ist. Deshalb wird im `require_once`-Befehl der relative Pfad zu addition4.inc.php angegeben. Pfad- und Dateinamen werden in Anführungszeichen und runden Klammern eingeschlossen

Die Extension .inc.php ist nicht zwingend notwendig, hat sich aber eingebürgert. .inc weist darauf hin, dass es sich um eine eingebettete Datei handelt (nicht alleine lauffähig). Die weitere Extension .php ist dann aus Sicherheitsgründen notwendig. Dateien mit der Extension .inc werden, wenn sie alleine aufgerufen werden, vom Server nicht ausgeführt. Errät ein Benutzer aber Pfad und Namen einer solchen Datei, kann er dies als URL direkt im Browser eingeben. Der Server führt den Code nicht aus, sondern überträgt ihn „as is“ an den Browser. Dort wird der Code dann dargestellt. Dies ist insbesondere dann peinlich, wenn der Code Passworte oder ähnliches enthält. Endet die Datei aber auf .php, wird sie auf jeden Fall vom Server ausgeführt.

Für die Auslagerung von php-Programmen in eigene Dateien sprechen im Prinzip die gleichen Argumente wie im JavaScript-Fall[3]. Allerdings sind hier auch Nachteile zu sehen. Mit php wird ein

[3]Bis auf das der geringeren Übertragungsmenge zum Client, da php ja auf dem Server ausgeführt wird.

Teil der html- Seite erzeugt, und die Auslagerung in eine eigene Datei kann eine Seite so auch unübersichtlicher machen.

Code, der von mehreren Seiten genutzt wird, sollte aber auf jeden Fall in eine eigene Datei ausgelagert werden.

Aufgabe 3.1 *Lagern Sie Ihren Code in den von Ihnen gelösten Aufgaben in eigenen Dateien aus!*

Kapitel 4

Grundsätzliches

4.1 Kommentare

Kommentare werden beim Programmablauf ignoriert. Kommentare sind dazu da, unseren Programmcode verständlicher zu machen. Um Kommentare richtig zu verstehen, betrachten wir Beispiel 4.1.

Beispiel 4.1 *Kommentare in JavaScript*

```
<!-- Das Programm addiert zwei einzugebende Zahlen
  Dateiname: addition2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition mit Eingabe </title>
</head>
<body>
    <script language = "JavaScript">
        /*  Autor: Bernd Bluemel
            Datum der letzten Aenderung: 17.09.01
            Zweck: Addition zweier Zahlen
            Dateiname: addition2.html
        */
        var ersterSummand; //nimmt den ersten Summanden auf
        var zweiterSummand; //nimmt den zweiten Summanden auf
        // Einlesen der Sumanden
        ersterSummand=prompt("Bitte geben Sie den ersten Summanden ein!","");
        zweiterSummand=prompt("Bitte geben Sie den zweiten Summanden ein!","");
        //Umwandeln der Summanden
        ersterSummand=parseFloat(ersterSummand);
        zweiterSummand=parseFloat(zweiterSummand);
        //Addieren
        summe=ersterSummand+zweiterSummand;
        //Ausgeben
        document.write ("Die Summe von " + ersterSummand +
                                " und " + zweiterSummand +
                                " ist: " + summe);
    </script>
</body>
</html>
```

In JavaScript und php gibt es zwei Arten von Kommentaren:

- Sämtlicher Text zwischen /* und */ wird ignoriert. In Beispiel 4.1 benutzte ich diese Kommentarform, um einige allgemeine Informationen zum Programm in die Datei aufzunehmen.
- Auf // folgender Text wird bis zum Zeilenende ignoriert.In Beispiel 4.1 benutzte ich diese Kommentarform, um den Sinn der einzelnen Programmzeilen zu dokumentieren.

Für dieses kleine Programm sind die Kommentare sicherlich unnötig. Doch schon bei geringfügig größeren Programmen sind Kommentare notwendig. Sie dienen für Sie dazu, Ihr eigenes Programm auch noch nach einigen Wochen zu verstehen. Anderen erleichtern Kommentare, sich in Ihre Gedanken hineinzuversetzen.

4.2 Variablen

Alle Programme, auch solche die im Browser laufen, manipulieren Daten, um gewünschte Ergebnisse zu erzielen.

Die Daten werden oftmals erst während des Programmlaufs eingelesen (z. B. die beiden Zahlen, die addiert werden sollen, in Beispiel 2.7).

Das Programm muss einen Namen haben für die Daten, mit denen es später arbeiten soll, damit die Dinge (später nennen wir Dinge Operationen), die mit den Daten gemacht werden sollen, beschrieben werden können.

Beispiel:

```
summe = ersterSummand + zweiterSummand
```

Hierbei ist

- *ersterSummand:* Der Name für die erste zu addierende Zahl (das erste einzulesende Datum).
- *zweiterSummand:* Der Name für die zweite zu addierende Zahl (das zweite einzulesende Datum)
- *summe:* Der Name für das Resultat (das auszugebende Datum).

Die Anweisung

```
summe = ersterSummand + zweiterSummand
```

ist nur möglich, weil die Daten, mit denen das Programm arbeiten soll, Namen bekommen haben.

Variablen sind die programminternen Namen für die Daten, mit denen das Programm arbeiten soll. Mit den Variablen kann dann beschrieben werden, was mit den Daten beim Programmlauf getan werden soll.

Grundsätzlich können auf Variablen unterschiedlichste Dinge abgespeichert werden, so z.B.:

- Zahlen (und hier auch noch verschiedene Zahlentypen wie ganze Zahlen, reelle Zahlen).
- Zeichenketten
- Kalenderdaten

Viele Programmiersprachen bieten die Möglichkeit, Variablen einen Datentyp zuzuweisen und damit festzulegen, welche Dinge auf dieser Variablen abgespeichert werden können. Hierdurch kann Speicherplatz gespart werden[1] und der Compiler kann weitaus besser prüfen, ob das eingegebene Programm Sinn macht.

JavaScript und php bieten diese Möglichkeit nicht. Solche Sprachen nennt man untypisiert.

Grundsätzlich unterstützen php und JavaScript aber folgende Datentypen:

- *Ganzzahlvariablen:* Auf Variablen diesen Typs können ganze Zahlen z. B. -1, 1000, 3, 7, 12, -1000) gespeichert werden.
- *Reelle oder Fließkommazahlen (Floats)*: Z.B. -1.2, -100.001, 3.1, 7.3, 1000.02)
  Merke: . (Punkt) statt , (Komma) in JavaScript und php als Dezimaltrenner.
- *Wahrheitswerte*: true oder false.
- *Strings:* Dies sind Zeichenketten z.B. „Bernd Blümel“, „Christiane Schäfer“.

In JavaScript werden Variablen mit dem Schlüsselwort `var` deklariert und können danach Werte eines beliebigen Datentyps annehmen. Der Typ einer Variablen kann im Programmverlauf wechseln. Variablen können beliebig im Programmcode deklariert werden. Variablen können bei der Deklaration initialisiert werden. Selbst die Deklaration einer Variablen ist in JavaScript eigentlich unnötig (vgl. Beispiel 4.2).

Beispiel 4.2 *Deklarationen, Initialisierungen und Zuweisungen in JavaScript*

```
<!-- Das Programm zur Variablendemonstration
             deklarationUndZuweisung.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Deklaration von Variablen in Javascript</title>
</head>
<body>
    <script language = "JavaScript">
        var x;      // x deklariert
        var y=19    // y deklariert, Wert 19 zugewiesen
                    // y ist Integer-Variable mit Wert 19
        document.write("Wert von y: " + y + "<BR>");
                    // y wird ausgegeben
                    // 19 erscheint im Browser
        x=4.7       // x den Wert 4.7 zugewiesen, x ist jetzt ein Float
        document.write("Wert von x: " + x + "<BR>");
                    // x wird ausgegeben
                    // 4.7 erscheint im Browser
        x="Datum"   // x ist jetzt String mit Wert "Datum"
        document.write("Neuer Wert von x: " + x + "<BR>");
                    // x wird ausgegeben
                    // Datum erscheint im Browser
        var i;      // i deklariert
        i="Test";   // i ist String mit Wert "Test"
        document.write("Wert von i: " + i + "<BR>");
                    // i wird ausgegeben
```

[1] Überlegen Sie sich, warum das so ist!

```
                    // Test erscheint im Browser
        z=4;        // z wurde nicht deklariert, dies ist in JavaScript nicht
                    //notwendig, z ist jetzt Integer-Variable mit Wert 4
        document.write("Wert von z: " + z + "<BR>");
                    // z wird ausgegeben
                    // 4 erscheint im Browser
    </script>
</body>
</html>
```

In php ist es sogar unmöglich, Variablen zu deklarieren. Man schreibt einfach den Namen einer Variablen in ein Programm und php erzeugt die Variable. Der Datentyp wird, wie bei JavaScript, automatisch zugeordnet. Beispiel 4.2 sieht also in php so aus:

Beispiel 4.3 *Deklarationen, Initialisierungen und Zuweisungen in php*

```
<!-- Das Programm zur Variablendemonstration
             deklarationUndZuweisung.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Deklaration von Variablen in php</title>
</head>
<body>
<?php
        // keine Deklarationen, gibt es in php nicht
        $y=19;       // y deklariert, Wert 19 zugewiesen
                    // y ist Integer-Variable mit Wert 19
        echo "Wert von y: $y <BR>";
                    // y wird ausgegeben
                    // 19 erscheint im Browser
        $x=4.7;      // x den Wert 4.7 zugewiesen, x ist jetzt ein Float
        echo "Wert von x: $x <BR>";
                    // x wird ausgegeben
                    // 4.7 erscheint im Browser
        $x="Datum";   // x ist jetzt String mit Wert "Datum"
        echo "Neuer Wert von x: $x <BR>";
                    // x wird ausgegeben
                    // Datum erscheint im Browser
        $i="Test";   // i ist String mit Wert "Test"
        echo "Wert von i: $i <BR>";
                    // i wird ausgegeben
                    // Test erscheint im Browser
        $z=4;       // ist Integer-Variable mit Wert 4
        echo "Wert von z:  $z <BR>";
                    // z wird ausgegeben
                    // 4 erscheint im Browser
?>
</body>
</html>
```

Beide Programme erzeugen dieselbe, in Abb. 4.1 dargestellte, Ausgabe.

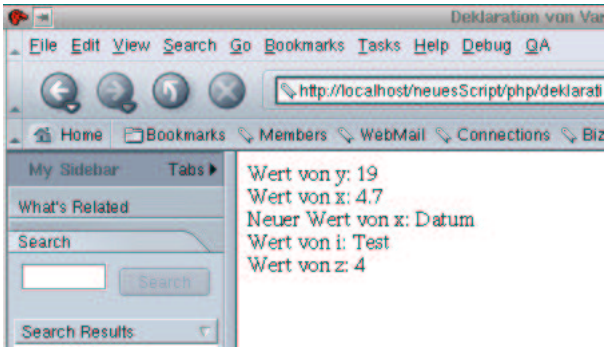


Abbildung 4.1: Ausgabe von Beispiel 4.3 bzw. 4.2

Meiner Ansicht nach ist diese Verhaltensweise der beiden Programmiersprachen ein echter Nachteil. Um dies zu verstehen, betrachten wir folgendes Beispiel:

Beispiel 4.4 *Schreibfehler führt zu fehlerhaftem Programm*

```
<!-- Das Programm zur Demonstration desastroeser
    Auswirkungen von Schreibfehlern bei
    Variablennamen
    Dateiname: schreibfehler.html      //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Schreibfehler bei Variablennamen </title>
</head>
<body>
    <script language = "JavaScript">
        eingabe = prompt ("Erste Eingabe", "");
        ersteEingabe = parseInt (eingabe);
        eingabe = prompt ("Zweite Eingabe", "");
        zweiteEingabe = parseInt (eingabe);
        ergebnis = ersteEingabe + zweiteEingabe;
        document.write("<B> Die Summe ist: " + ergebnis + "</B><BR>");
        ergbnis = ersteEingabe * zweiteEingabe;
        //Schreibfehler ergbnis statt ergebnis
        document.write("<B> Das Produkt ist: " + ergebnis + "</B><BR>");
        // die Summe wird zum zweiten Mal ausgegeben
        // das Programm ist fehlerhaft
    </script>
</body>
</html>
```

Abb. 4.2 zeigt meine Eingaben und die Ausgaben des kurzen Programms. Die erste Ergebniszeile ist schon mal gut. Ich wandle die auf die Variable eingabe eingelesenen Strings durch `parseInt` in Integers um und weise das Ergebnis auf ersteEingabe bzw. zweiteEingabe zu. Damit funktioniert die Addition und das erste Ergebnis ist richtig. Die zweite Ergebniszeile dann weniger.

Auch das Produkt ist 9. Nun ist aber 4*5 definitiv 20 und nicht 9. Des Rätsels Lösung ist die Zeile:

```
ergbnis = ersteEingabe * zweiteEingabe;
```

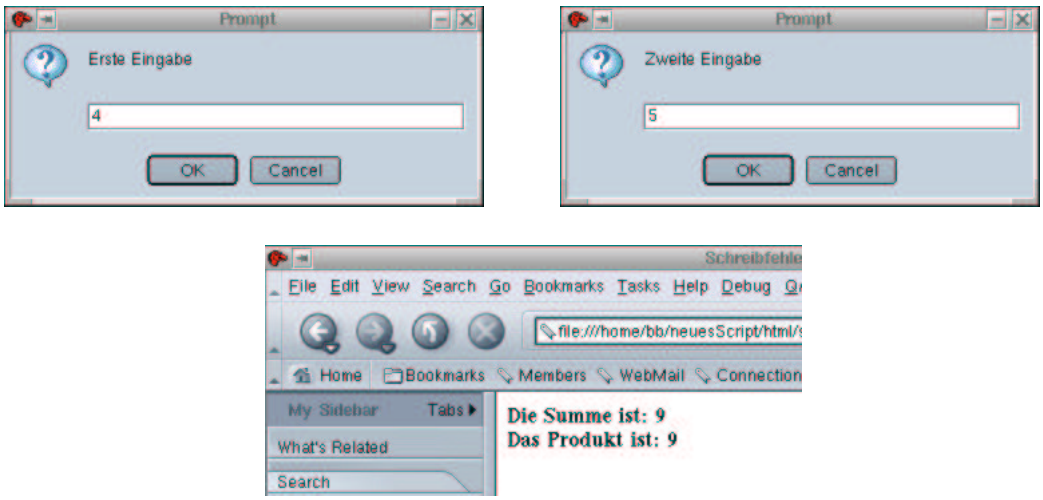


Abbildung 4.2: Beispielhafte Ein-und Ausgabe von Beispiel **??**

In dieser Zeile ist mir ein Schreibfehler unterlaufen (ergbnis statt ergebnis). Und JavaScript erzeugt einfach eine weitere Variable namens ergbnis und weist das Ergebnis der Multiplikation dieser Variablen zu. Ausgegeben wird aber wieder die Variable ergebnis:

```
document.write("<B> Das Produkt ist: " + ergebnis + "</B><BR>");
```

Dadurch, dass Variablen nicht deklariert werden müssen, können Schreibfehler also recht drastische Auswirkungen haben.

Das Perfide an diesen Fehlern ist, dass der Computer bzw. die JavaScript- oder php-Umgebung diese Fehler nicht finden kann. Die Programme scheinen zu laufen, nur die Ergebnisse sind falsch. Besser sind immer Fehler[2], die von der Programmierumgebung selbst erkannt werden können.

Ich demonstriere dies in Beispiel 4.5.

Beispiel 4.5 *Von JavaScript erkannter Schreibfehler*

```
<!-- Das Programm zur Demonstration
    erkannter Fehler
    Dateiname: schreibfehler2.html      //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Schreibfehler bei Variablennamen </title>
</head>
<body>
    <script language = "JavaScript">
        eingabe = prompt ("Erste Eingabe", "");
        ersteEingabe = parseInt (eingabe);
        eingabe = prompt ("Zweite Eingabe", "");
        zweiteEingabe = parseInt (eingabe);
        ergebnis = ersteEingabe + zweiteEingabe;
        document.write("<B> Die Summe ist: " + ergebnis + "</B><BR>");
        ergbnis = ersteEingabe * zweiteEingabe;
```

[2] Soweit Fehler überhaupt gut sein können.

```
        //Schreibfehler ergbnis statt ergebnis
        document.write("<B> Das Produkt ist: " + ergebnis + "</B><BR>);
        // die Summe wird zum zweiten Mal ausgegeben
        // das Programm ist fehlerhaft
    </script>
</body>
</html>
```

Dort fehlen in der Zeile

```
document.write("<B> Das Produkt ist: " + ergebnis + "</B><BR>);
```

die schließenden Anführungszeichen. JavaScript erkennt den Fehler und teilt uns in einem Fehlerfenster mit, welche Art Fehler aufgetreten ist (vgl. Abb. 4.3).

Abbildung 4.3: Fehlermeldung von Beispiel 4.5

In anderen Programmiersprachen (z.B. Java, C, C++) müssen Variablen vor der Benutzung deklariert werden. Dadurch werden in diesen Programmiersprachen Fehler wie in Beispiel 4.4 erkannt. In JavaScript und php ist dies nicht möglich.

Abschließend noch einige Regeln zu Variablennamen:
Merke: Diese Regeln gelten auch für alle anderen Namen, die Sie selbst vergeben können[3] (Ausnahme: Nur Variablennamen beginnen in php mit einem $).

- Variablennamen beginnen in php mit dem $, in JavaScript mit einem Buchstaben. In php muss auf das $-Zeichen ein Buchstabe folgen.
- Variablennamen können nach dem ersten Buchstaben Ziffern und Buchstaben in beliebiger Reihenfolge enthalten.
- Variablennamen dürfen außer dem _ keine Sonderzeichen enthalten. Dies schließt auch deutsche Umlaute mit ein.
- Variablennamen können beliebig lang sein.
- Groß- und Kleinschreibung spielt eine Rolle (summe $\neq$ Summe $\neq$ suMME $\neq$ SUMME).
- In JavaScript bzw. php reservierte Worte[4] dürfen keine Variablennamen sein.
- *WICHTIG: Variablennamen sollten einen Bezug zu den Daten haben, die sie später aufnehmen sollen.*

[3] Wo Sie noch Namen vergeben können, werden Sie später lernen
[4] Was reservierte Worte sind, werden Sie auch später lernen!

4.3 Ausdrücke

Ein Ausdruck ist eine Regel zur Berechnung eines Wertes Das klingt kompliziert, kennen wir aber schon aus unseren vielen Beispielen. Die Summenbildung, die wir ja aus so vielen Beispielen kennen, ist ein typischer Ausdruck.

Ein Ausdruck besteht aus mehreren (oder auch nur einem) Operanden und mehreren (oder auch nur einem) Operator. Abb. 4.4 zeigt Operanden und Operator unseres Summenbeispiels. Ein Operand

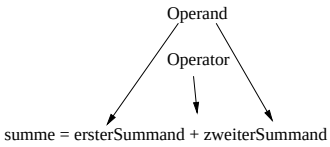


Abbildung 4.4: Ausdruck mit Operanden und Operator

kann sein:

- Eine Variable (richtiger: ihr Inhalt).
- Eine Konstante (bruttobetrag = 1.16 * nettobetrag, 1.16 ist ein konstanter Wert im Programm und wird daher als Konstante bezeichnet.).
- Ein Funktionsaufruf (-¿ später).

4.4 Operatoren

4.4.1 Der Zuweisungsoperator

Einer der wichtigsten Operatoren ist der Zuweisungsoperator. Die Zuweisung (oder der Zuwei- sungs-operator) ist einfach das Gleichheitszeichen. Wir haben den Zuweisungsoperator implizit in allen unseren Beispielen genutzt. Betrachten wir wieder unsere Summation:

```
summe = ersterSummand + zweiterSummand
```

Was passiert hier? Der Ausdruck auf der rechten Seite (`ersterSummand + zweiterSummand`) wird ausgewertet (berechnet) und das Ergebnis wird der Variablen auf der linken Seite `summe` zugewiesen.

Daher muss auf der rechten Seite des Zuweisungsoperators immer ein Ausdruck und auf der linken Seite eine Variable stehen.

Merke: Der Ergibt-Operator (anderer Name für das Gleichheitszeichen) überschreibt. Das bedeutet in unserem Beispiel, dass der vorherige Wert von summe durch den neuen Wert (den berechneten Wert) ersetzt wird.

Merke: Variablen können daher im Programmablauf verschiedene Werte haben. Dies zeigt sich auch sehr schön an der Wertetabelle in Tabelle 2.1.

4.4.2 Arithmetische Operatoren

php und JavaScript kennen und unterstützen die gängigen arithmetischen Operatoren. Wie auch in der Schulmathematik, besitzen diese Operatoren in php und JavaScript verschiedenen Vorrang (Punkt-

geht vor Strichrechnung). Wie auch in der normalen Mathematik, kann man den Vorrang der Operatoren durch Klammerung ändern.

In Tabelle 4.1 sind die arithmetischen Operatoren und ihr Vorrang (Reihenfolge) aufgelistet.

Operator	Operation	Vorrang
`()`	Klammern	werden zuerst ausgewertet. Sind Klammern von Klammern umschlossen, werden sie von innen nach außen ausgewertet.
`*, /, %`	Multiplikation, Division, Modulus	werden als zweites ausgewertet
`+, -`	Addition, Subtraktion	werden zuletzt ausgewertet

Tabelle 4.1: Der Vorrang der arithmetischen Operatoren in JavaScript und php. Mehrere Operatoren gleichen Vorrangs werden stets von links nach rechts ausgewertet.

Beispiel 4.6 zeigt ein JavaScript-Beispiel, Abb. 4.5 den Output. Die php-Implementierung ist analog.

Beispiel 4.6 *Arithmetische Operatoren*

```
<!-- Das Programm zur Demonstration arithmetischer Operatoren
  Dateiname: arithmetischeOperatoren.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Arithmetische Operatoren</title>
</head>
<body>
    <script language = "JavaScript">
        var i = 2;
        var j = 3;
        var l;
        var r= 2;
        var s=3;
        var t;
        l=i+j; // 5;
        document.write("l1: " + l + "<br>");
        l=i-j; // -1
        document.write("l2: " + l + "<br>");
        l=j%i; // 1 (Modulo-Bildung)
        document.write("l3: " + l + "<br>");
        l=i*j; // 6
        document.write("l4: " + l + "<br>");
        t=s/r; // 1.5 Division
        document.write("t: " + t + "<br>");
        l=r+i*j; // 8
        document.write("l4: " + l + "<br>");
        l=(r+i)*j;  // 12
        document.write("l5: " + l + "<br>");
    </script>
</body>
</html>
```

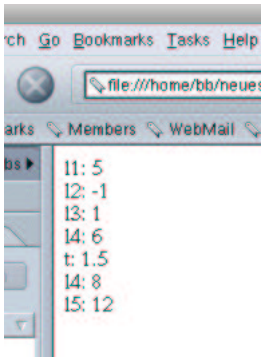


Abbildung 4.5: Arithmetische Operatoren

Wie alle C-ähnlichen Sprachen erlauben JavaScript und php in einigen Fällen abkürzende Schreibweisen.

```
i += j // i = i + j
i -= j // i = i -j
i *= j // i = i * j
i /= j // i = i / j
i \%= j // i = i \% j
i++ //i=i+1
i-- //i=i-1
```

In JavaScript zeigt der Operator +, auf Strings angewendet, ein Sonderverhalten. Er fügt zwei Strings zu einem Ergebnisstring zusammen. Beispiel 4.7 und Abb. 4.6 zeigen dies.

Beispiel 4.7 *Der Plus-Operator bei JavaScript*

```
 <!-- Das Programm zur Demonstration arithmetischer Operatoren
  Dateiname: plusOperator.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Der +-Operator bei Strings</title>
</head>
<body>
    <script language = "JavaScript">
        var s1="ab";
        var s2="cd";
        var ergebnis;
        var i=5;
        var j="6";
        ergebnis=s1+s2;
        document.write("ergebnis1: " + ergebnis + "<br>");
        ergebnis=s1+i;
        document.write("ergebnis2: " + ergebnis + "<br>");
        ergebnis=j+i;
        document.write("ergebnis3: " + ergebnis + "<br>");
        j=parseInt(j);
        ergebnis=j+i;
```

```
        document.write("ergebnis4: " + ergebnis + "<br>");
    </script>
</body>
</html>
```

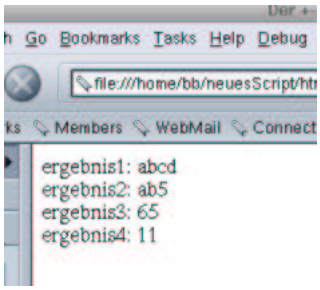


Abbildung 4.6: Der Plus-Operator in JavaScript

In php ist dies nicht der Fall. php benutzt den Punkt(.), um zwei Strings miteinander zu verbinden. Beispiel 4.8 und Abb. 4.7 zeigen dies.

Beispiel 4.8 *Der Plus-Operator und der Punkt-Operator bei php*

```
 <!-- Das Programm zur Demonstration arithmetischer Operatoren
  Dateiname: plusOperator.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Der +-Operator bei Strings</title>
</head>
<body>
<?php
        $s1="ab";
        $s2="cd";
        $i=5;
        $j="6";
        $ergebnis=$s1+$s2;
        echo "ergebnis1: $ergebnis <br>";
        $ergebnis=$s1.$s2;
        echo "ergebnis2: $ergebnis <br>";
        $ergebnis=$s1+$i;
        echo "ergebnis3: $ergebnis <br>";
        $ergebnis=$s1.$i;
        echo "ergebnis4: $ergebnis <br>";
        $ergebnis=$j.$i;
        echo "ergebnis5: $ergebnis <br>";
        $ergebnis=$j+$i;
        echo "ergebnis6: $ergebnis <br>";
?>
</body>
</html>
```

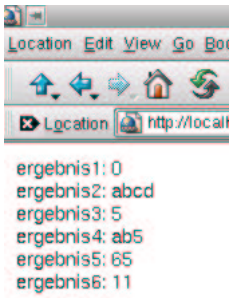


Abbildung 4.7: Ausgabe von Beispiel 4.8

Wir sehen aber, dass auch php eigentlich unschöne Dinge tut. Strings, die sich nicht in Zahlen umwandeln lassen, werden bei der Addition auf 0 gesetzt. Eine Fehlermeldung, dass diese Operation mit den beteiligten Operanden wenig Sinn macht, wäre m.E. angebrachter.

4.4.3 Vergleichsoperatoren

Tabelle 4.2 zeigt die in JavaScript und php verfügbaren Vergleichsoperatoren.

Algebraischer Operator	Operator in JavaScript/php	Beispiel	Bedeutung
=	==	x == y	x ist gleich y
≠	!=	x != y	x ist ungleich y
>	>	x > y	x ist größer als y
<	<	x < y	x ist kleiner als y
≥	>=	x >= y	x ist größer gleich y
≤	<=	x <= y	x ist kleiner gleich y

Tabelle 4.2: Vergleichsoperatoren in JavaScript und php

Vergleichsoperatoren werden eigentlich immer in Verbindung mit Kontrollstrukturen und Schleifen genutzt. Beide Programmierkonstrukte wurden bisher noch nicht behandelt. Darum wirken die jetzt vorgestellten Beispiele etwas konstruiert. Um die Ergebnisse der Vergleiche zu visualisieren, benutze ich implizit Variablen vom Typ Boolean. Solche Variablen können nur zwei Werte annehmen: true oder false.

Besondere Vorsicht ist bei dem Operator == geboten. Er darf keinesfalls mit dem Zuweisungsoperator (=) verwechselt werden. Beispiel 4.9 und Abb. 4.8 zeigen einige Vergleiche und deren Ergebnisse.

Beispiel 4.9 *Vergleichsoperatoren in JavaScript*

```
<!-- Das Programm zur Demonstration von Vergleichsoperatoren
  Dateiname: vergleichsOperatoren.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Vergleichsoperatoren</title>
```

```
</head>
<body>
    <script language = "JavaScript">
        var s1="ab";
        var s2="cd";
        var i=5;
        var j="5";
        ergebnis=(s1==s2);
        document.write("ergebnis1: " + ergebnis + "<br>");
        ergebnis=(s1=s2);
        document.write("ergebnis2: " + ergebnis + " s1: " + s1 +  "<br>");
        ergebnis=(s1!=i);
        document.write("ergebnis3: " + ergebnis + "<br>");
        ergebnis=(s1>i);
        document.write("ergebnis4: " + ergebnis + "<br>");
        ergebnis=(j==i);
        document.write("ergebnis5: " + ergebnis + "<br>");
        ergebnis=(j=i);
        document.write("ergebnis6: " + ergebnis + "<br>");
    </script>
</body>
</html>
```

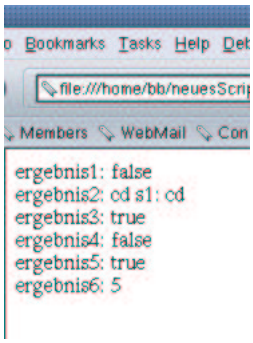


Abbildung 4.8: Ausgabe von Beispiel 4.9

Zeilen, wie

```
ergebnis=(s1>i)
```

haben wir in dieser Form noch nicht besprochen. Dennoch ist auch dies nur ein einfacher Ausdruck. Links steht eine Variable. Rechts sehen wir zwei weitere Variablen und zwischen ihnen den Größer-Operator.

Was nun passiert, ist recht einfach. JavaScript führt den Vergleich durch. Ist das Ergebnis wahr, wird der Variable true zugewiesen, stimmt der Vergleich nicht, false.

Ansonsten sehen wir, dass sich Vergleichsoperatoren so verhalten, wie wir das erwarten.

Eine Ausnahme besteht, wenn wir den Zuweisungsoperator verwenden. Das Ergebnis einer Zuweisung ist der Wert der zugewiesenen Variable[5]. Dies ist insbesondere, wenn wir uns im nächsten Kapitel mit Kontrollstrukturen (der if-Anweisung) beschäftigen, von besonderer Bedeutung.

[5]Dieses absonderliche Verhalten liegt einfach daran, dass eine Zuweisung kein Vergleich ist!

php verhält sich ähnlich, allerdings wird true in php durch die 1 ersetzt, false durch 0.

4.4.4 Logische Operatoren

Logische Operatoren werden hauptsächlich genutzt, um Vergleiche zu verketten oder zu negieren.

Logisches UND

Der && Operator führt die logische und-Verknüpfung durch. Er ergibt true, wenn seine beiden Operanden true ergeben, ansonsten false.

Logisches ODER

Der || Operator führt die logische oder-Verknüpfung durch. Er ergibt true, wenn einer seiner beiden Operanden true ergibt, ansonsten false.

Logische Negation

Der Negations-Operator ! wird auf nur einen Operanden angewendet. Er ergibt true, wenn sein Operand false ist und umgekehrt. Beispiel 4.10 und Abb. 4.9 zeigen einige durch logische Operatoren verkettete Vergleiche und deren Ergebnisse.

Beispiel 4.10 *Logische Operatoren in JavaScript*

```
<!-- Programm zur Demonstration von logischen Operatoren
  Dateiname: logischeOperatoren.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Logische Operatoren</title>
</head>
<body>
    <script language = "JavaScript">
        var i=5;
        var j=6;
        var k=6;
        var l=5;
        ergebnis = (i==j) && (j==k) // Wert von ergebnis: false, da i nicht j
        document.write("ergebnis1: " + ergebnis + "<br>");
        ergebnis = (i==j) || (j==k) // Wert von ergebnis: true, da j gleich k
        document.write("ergebnis2: " + ergebnis + "<br>");
        ergebnis = (i==l) && (j==k) // Wert von ergebnis: true
        document.write("ergebnis3: " + ergebnis + "<br>");
        ergebnis = (i==l) || (j==k) // Wert von ergebnis: true
        document.write("ergebnis4: " + ergebnis + "<br>");
        ergebnis = !(i==j)  // Wert von ergebnis: true
        document.write("ergebnis5: " + ergebnis + "<br>");
        ergebnis = !(i!=j)  // Wert von ergebnis: false
        document.write("ergebnis6: " + ergebnis + "<br>");
    </script>
</body>
</html>
```

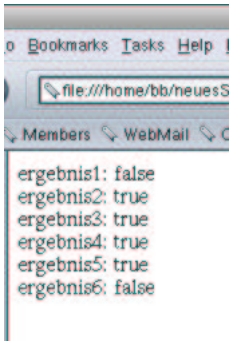


Abbildung 4.9: Ausgabe von Beispiel 4.10

Tabelle 4.3: Die and-Wertetabelle

a/b	true	false
true	true	false
false	false	false

Die Zeile

```
ergebnis = (i==j) && (j==k)
```

ähnelt den Ausdrücken des vorherigen Kapitels stark. Der einzige Unterschied besteht darin, dass wir auf der rechten Seite des Ausdrucks zwei Vergleiche sehen. Beide Vergleiche sind durch einen logischen Operator (in diesem Beispiel &&) verbunden. JavaScript wertet zunächst beide logischen Ausdrücke aus und danach die Verbindung. Das Ergebnis wird der Variable ergebnis zugewiesen.

Tabellen 4.3 bis 4.5 zeigen die möglichen Ergebnisse der logischen Operatoren. Tabelle 4.3 ist folgendermaßen zu lesen: Wenn a den Wert true besitzt und b ebenfalls, dann ist a and b true. Wenn a true ist und b false, dann ist a and b false. Die weiteren Tabellen sind analog zu lesen.

4.4.5 Der tenäre Operator

Der tenäre Operator hat die Form `?:`. Der tenäre Operator erhält 3 Operanden. Der erste kommt vor das Fragezeichen, der zweite zwischen Fragezeichen und Doppelpunkt, der dritte hinter den Doppelpunkt. Der erste Operand muss einen bool'schen Wert ergeben, also true oder false. Er ist gewöhnlich ein Vergleich. Ergibt der erste Operand true, wird das Ergebnis des zweiten Operanden zurückgegeben, ansonsten das des dritten Operanden (vgl. Beispiel 4.11 und Abb. 4.10).

Beispiel 4.11 *Der tenäre Operator*

```
<!-- programm zur Demonstration des tenaeren Operators
```

Tabelle 4.4: Die or-Wertetabelle

a/b	true	false
true	true	true
false	true	false

Tabelle 4.5: Die not-Wertetabelle

a	not a
true	false
false	true

```
  Dateiname: tenaer.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Ten&auml;rer Operator</title>
</head>
<body>
    <script language = "JavaScript">
        var i = 1;
        var j = 2;
        var k;
        (i == j)? k = i : k = 6;
        document.write ("k hat den Wert: " + k);
    </script>
</body>
</html>
```

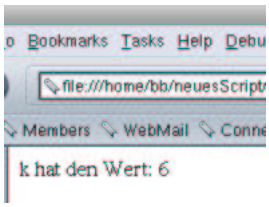


Abbildung 4.10: Ausgabe von Beispiel 4.11

4.5 Konstante

Programme benötigen häufig konstante Werte (im folgenden Konstanten genannt), um Berechnungen durchführen zu können.

Eine Konstante kann explizit durch „Hinschreiben“ an jeder gewünschten Stelle des Programms „definiert“ werden. Dies zeigt Beispiel 4.12.

Beispiel 4.12 *Konstante Werte in JavaScript*

```
<!-- Das Programm demonstriert die Notwendigkeit von Konstanten
  Dateiname: konstante1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
```

```
  <title>Konstante</title>
</head>
<body>
    <script language = "JavaScript">
        var einkaufspreis;
        var nettoVerkaufspreis;
        var bruttoVerkaufspreis;
        einkaufspreis=prompt ("Bitte geben Sie den Einkaufspreis ein!","");
        nettoVerkaufspreis=1.16*einkaufspreis;
        bruttoVerkaufspreis=1.16*nettoVerkaufspreis;
        document.write("Netto-Verkaufspreis: " + nettoVerkaufspreis + "<br>" +
                       "Brutto-Verkaufspreis: " + bruttoVerkaufspreis);
    </script>
</body>
</html>
```

Dies ist nicht immer sinnvoll:

- Die Zahl 1.16 hat geringere Aussagekraft, als das Wort Mehrwertsteuersatz.
- Die Konstante 1.16 hat in diesem Beispiel zwei Bedeutungen:
  - Gewinnspanne.
  - Mehrwertsteuersatz.

Das macht des Programm schwerer verständlich.

Ändert sich der Mehrwertsteuersatz, kann ich in Beispiel 4.12 nicht einmal mit der Funktion „Suchen und Ersetzen“ den Mehrwertsteuersatz ändern. Ich würde die Gewinnspanne auch erhöhen.

Nehmen wir an, ein Programm benötigt öfter den Mehrwertsteuersatz. Eine (syntaktisch richtige) Lösung wäre, auch in diesem Fall den Mehrwertsteuersatz an jeder Stelle des Programms, wo er benötigt wird, hinzuschreiben.

Ändert der Gesetzgeber den Mehrwertsteuersatz, muss jede Stelle des Programms, wo der Mehrwertsteuersatz vorkommt, geändert werden.

Eigentlich alle Programmiersprachen erlauben es, Konstanten mit Namen zu definieren und diesen einmalig einen Wert zuzuweisen. In php ist dies auch möglich, JavaScript bildet eine Ausnahme.

In php werden Konstanten mit der Syntax

```
define("konstantenname", "wert");
```

gebildet. Ich demonstriere dies an einem Beispiel:

Beispiel 4.13 *Konstante in php*

```
<!-- Programm zur Konstantendemonstration
    Dies ist die Eingabemaske
  Dateiname: konstante.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Konstante</title>
</head>
<body>
<h2>
```

```
    Konstantendemonstration
</h2>
<form name="konstante1" action="./konstante.php" method="post">
    <table border>
        <tr>
            <td>
                Einkaufspreis
            </td>
            <td>
                <input type="text" name="einkaufspreis" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Beispiel 4.14 *Konstante Werte in php 2*

```
  <!-- Programm zur Konstantendemonstration
    Die Logik
  Dateiname: konstante.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Konstantendemonstration</title>
</head>
<body>
<h2>
    Konstantendemonstration
</h2>
<?php
    define("MEHRWERSTEUERSATZ","1.16");
    define("GEWINNSPANNE","1.16");
    $nettoVerkaufspreis=GEWINNSPANNE*$einkaufspreis;
    $bruttoVerkaufspreis=MEHRWERSTEUERSATZ*$nettoVerkaufspreis;
    echo "Der Bruttoverkaufspreis betr&auml;gt: $bruttoVerkaufspreis!";
?>
</body>
</html>
```

Änderungen des Mehrwertsteuersatzes implementiere ich in meinem Programm, indem ich einfach die Zeile:

```
define("MEHRWERSTEUERSATZ","1.16");
```

ändere[6]. Konstanten unterscheiden sich von Variablen dadurch, dass sich ihr Wert während des Programmlaufs nicht ändern kann (Konstanten sind halt konstant). Konstanten dürfen daher nie auf der

[6] Wie wird wohl die Gewinnspanne geändert?

linken Seite einer Zuweisung stehen. Beachten Sie: Konstanten in php beginnen *nicht* mit dem $-Zeichen!!

Konstanten werden häufig, obwohl dies nicht vorgeschrieben ist, mit Großbuchstaben benannt, um sie „in das Auge springend“ von Variablen zu unterscheiden.

JavaScript kennt, wie bereits erwähnt, keine Konstante. Aber auch hier kann ich Abhilfe schaffen, indem ich einfach einer Variablen den konstanten Wert zuweise und diesen in meinem Programm nicht mehr ändere[7] (vgl. Beispiel 4.15).

Beispiel 4.15 *Konstante Werte in JavaScript (richtig)*

```
<!-- Das Programm demonstriert die Notwendigkeit von Konstanten
  Dateiname: konstante1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Konstante</title>
</head>
<body>
    <script language = "JavaScript">
        var einkaufspreis;
        var nettoVerkaufspreis;
        var bruttoVerkaufspreis;
        var mehrwertsteuer=1.16;
        var gewinnspanne=1.16;
        einkaufspreis=prompt ("Bitte geben Sie den Einkaufspreis ein!","");
        nettoVerkaufspreis=gewinnspanne*einkaufspreis;
        bruttoVerkaufspreis=mehrwertsteuer*nettoVerkaufspreis;
        document.write("Netto-Verkaufspreis: " + nettoVerkaufspreis + "<br>" +
                       "Brutto-Verkaufspreis: " + bruttoVerkaufspreis);
    </script>
</body>
</html>
```

Änderungen der Mehrwertsteuer führe ich hier durch Änderung der Zeile

```
var mehrwertsteuer=1.16;
```

durch.

Aufgabe 4.1 *Fügen Sie Kommentare in Ihre bisherigen Programme ein!*

Aufgabe 4.2 *Einige der Beispiele dieses Kapitels zeigen nur JavaScript-Lösungen. Entwickeln Sie die entsprehenden php-Lösungen!*

Aufgabe 4.3 *Verifizieren Sie Tabelle 4.3 bis Tabelle 4.5!*

Aufgabe 4.4 *In vielen Beispielen dieses Kapitels waren die Variablennamen nicht gerade sprechend. Warum ist dies kein Widerspruch zu der Aussage, dass Variablennamen Aussagen, über die Werte, die auf ihnen abgespeichert werden, zulassen sollen?*

[7] Das kann man natürlich auch in php so machen, doch die Definition einer Konstanten zeigt anderen Programmierern (und auch mir später), was ich vorhatte.

Kapitel 5

Beispiele

In diesem Kapitel wollen wir den bisher dargestellten Stoff an einigen Beispielen vertiefen. Diese Beispiele wollen wir dann im weiteren Verlauf ausbauen.

5.1 Euro-Dollar-Umrechnung

Diese Anwendung soll es den Anwendern ermöglichen, Euro-Beträge zum jeweils gültigen Kurs in Dollar umrechnen zu lassen. Eingegeben wird ein Euro-Betrag, ausgegeben der umgerechnete Dollar-Betrag.

Beispiel 5.1 *Euro-Dollar-Umrechnung Teil1*

```
<!-- Euro-Dollar-Umrechnung Teil1
  Dateiname: euro1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 1</title>
</head>
<body>
    <script language = "JavaScript">
        var euroBetrag;
        var dollarBetrag;
        var kurs=0.9;
        eurobetrag=prompt("Bitte geben Sie den Eurobetrag ein!");
        eurobetrag=parseFloat(eurobetrag);
        dollarbetrag=kurs*eurobetrag;
        document.write(eurobetrag + " Euro entsprechen zur Zeit " +
                        dollarbetrag + " Dollar.");
    </script>
</body>
</html>
```

Wir lesen zunächst den Eurokurs mittels `prompt` ein, wandeln in einen `float` um und multiplizieren den erhaltenen Wert mit dem Euro-Kurs. Das Ergebnis wird mit `document.write` ausgegeben.

Abb. 5.1 zeigt eine beispielhafte Anwendung des Programms.

Die Implementierung dieses Beispiels in php zeigen die Beispiele 5.2 und 5.3. Eine beispielhafte Anwendung der php-Lösung findet sich in Abb. 5.2. Den Code müssten Sie eigentlich ohne weitere Erläuterung verstehen.

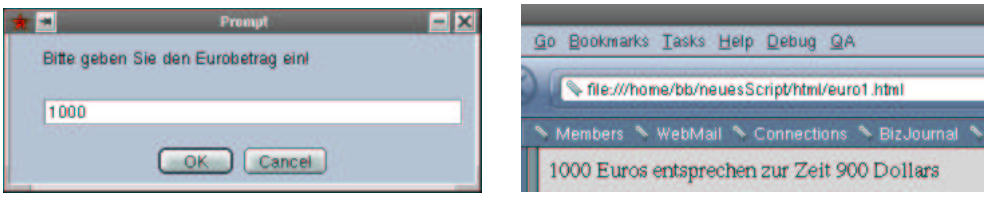


Abbildung 5.1: Das Euro-Dollar Beispiel Teil 1

Beispiel 5.2 *Euro-Dollar-Umrechnung Teil 1 in php (html-Datei)*

```
<!-- Programm zur Euro-Dollar Umrechnung
    Dies ist die Eingabemaske
  Dateiname: euro1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung</title>
</head>
<body>
<h2>
    Euro-Dollar Umrechnung
</h2>
<form name="euro1" action="./euro1.php" method="post">
    <table border>
        <tr>
            <td>
                Euros
            </td>
            <td>
                <input type="text" name="eurobetrag" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Beispiel 5.3 *Euro-Dollar-Umrechnung Teil 1 in php (php-Datei)*

```
  <!-- Programm zur Euro-Dollar Umrechnung
    Die Logik
  Dateiname: euro1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung</title>
```

```
</head>
<body>
<h2>
    Euro-Dollar Umrechnung: Das Ergebnis
</h2>
<?php
  $kurs=0.9;
  $dollarbetrag=$kurs*$eurobetrag;
  echo "$eurobetrag Euro entspricht $dollarbetrag Dollar!";
?>
</body>
</html>
```

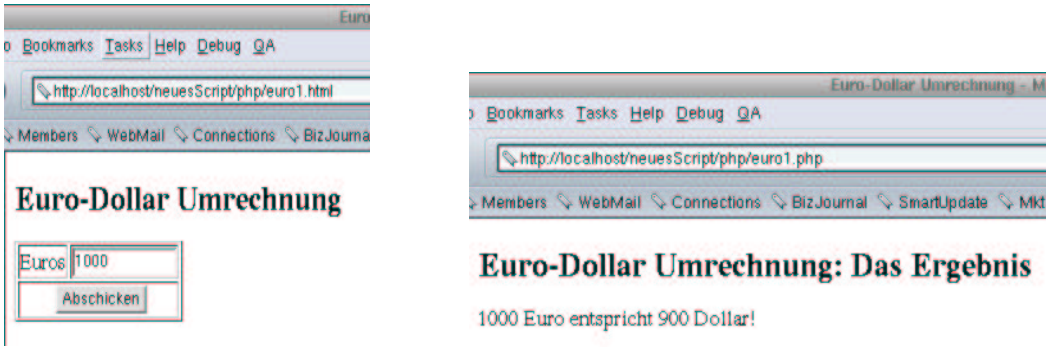


Abbildung 5.2: Das Euro-Dollar Beispiel Teil 1 in php

5.2 Volatilität

Die zu erstellende Anwendung soll die Volatilität von Aktien berechnen. Die Volatilität misst das Risiko einer Aktie; je höher sie ist, desto unsicherer ist der zukünftige Kurs der Aktie.

Eingegeben werden soll der aktuelle, der gestrige und der vorgestrige Kurs der Aktie. Die Anwendung soll den Durchschnittskurs und die Volatilität ausgeben. Zur Berechnung der Volatilität gelten folgende Rechenregeln:

- Der Durchschnitt $\bar{x}$ der drei Kurswerte x_1, x_2, x_3 lautet mit $n = 3$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \text{also} \quad \boxed{\bar{x} = (x_1 + x_2 + x_3)/3.}$$

- Die *Volatilität* vol ergibt sich aus drei Rechenschritten:

  1. Errechne die *Varianz* var für die $n = 3$ Kurswerte,

$$\text{var} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2, \qquad \text{also} \quad \boxed{\text{var} = [(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2]/2.}$$

  (*Hinweis:* Programmieren Sie die Quadratfunktion elementar, also x^2 als $x \cdot x$.)

2. Berechne die Konstante dt als Quotient der $n = 3$ eingegebenen Börsentage durch die 250 Börsentage eines Jahres:

$$dt = n/250, \qquad \text{also} \quad \boxed{dt = 3/250.}$$

3. Daraus ergibt sich die Volatilität vol gemäß

$$\boxed{\text{vol} = \text{var}/\sqrt{dt}}.$$

(*Hinweis:* Die Wurzelfunktion ist in JavaScript durch `Math.sqrt()` und in php durch `sqrt()` gegeben: $\sqrt{x}$ ist also `Math.sqrt(x)` bzw. `sqrt(x)`.)

Obwohl die Aufgabenstellung zunächst schwierig erscheint, ist die Lösung doch recht einfach. Wir müssen:

- Drei Zahlen einlesen.
- Die Zahlen in `float` konvertieren (nur JavaScript).
- Die Formeln programmieren.
- Die Ergebnisse ausgeben.

Das Programmieren der Formeln ist aber auch einfach. Da sowohl JavaScript als auch php über arithmetische Operatoren verfügen, können wir die Formeln mit geeigneter Klammerung direkt in die Programme übernehmen. Das einzige Problem ist die Anwendung der Funktionen `Math.sqrt(x)` bzw. `sqrt(x)`[1]. Das aber ist auch nicht wirklich schwer, weil Sie seit einiger Zeit schon, ohne es zu wissen, mit Funktionen arbeiten[2].

Beispiel 5.4 zeigt die Realisierung in JavaScript, Beispiele 5.5 und 5.6 die php-Implementierung. Abb. 5.3 gibt einen beispielhaften Durchlauf durch Beispiel 5.7 wieder, Abb. 5.4 zeigt selbiges für php[3].

Beispiel 5.4 *Volatilitäten Teil 1*

```
<!--Volatilitaeten Teil1
  Dateiname: volatilitaeten1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Volatilit&auml;ten </title>
</head>
<body>
    <script language = "JavaScript">
        var kursHeute;
        var kursGestern;
        var kursVorgestern;
        var durchschnitt;
        var varianz;
        kursHeute=prompt("Bitte geben Sie den heutigen Kurs ein!");
        kursGestern=prompt("Bitte geben Sie den gestrigen Kurs ein!");
```

[1] Weil Sie ja noch gar nicht wissen, was Funktionen sind, das lernen Sie erst in Kapitel —

[2] `parseFloat` ist z.B. eine Funktion

[3] Glücklicherweise sind bei gleichen Eingaben auch die Ergebnisse gleich!

```
        kursVorgestern=prompt("Bitte geben Sie den vorgestrigen Kurs ein!");
        kursHeute=parseFloat(kursHeute);
        kursGestern=parseFloat(kursGestern);
        kursVorgestern=parseFloat(kursVorgestern);
        // Durchschnittsberechnung
        durchschnitt=(kursHeute+kursGestern+kursVorgestern)/3;
        // Varianz
        varianz=((kursHeute-durchschnitt)*(kursHeute-durchschnitt) +
                        (kursGestern-durchschnitt)*(kursGestern-durchschnitt) +
                        (kursVorgestern-durchschnitt)*(kursVorgestern-durchschnitt))/
        dt=3/250;
        //Volatilitaet
        volatilitaet=varianz/Math.sqrt(dt);
        document.write("Der Duchschnitt der eingegebenen Werte ist: " +
                            durchschnitt + "<br>" +
                            "Die Varianz der eingegebenen Werte ist: " +
                            varianz + "<br>" +
                            "Die Volatilit&auml;t der eingegebenen Werte ist: " +
                             volatilitaet + "<br>");
    </script>
</body>
</html>
```

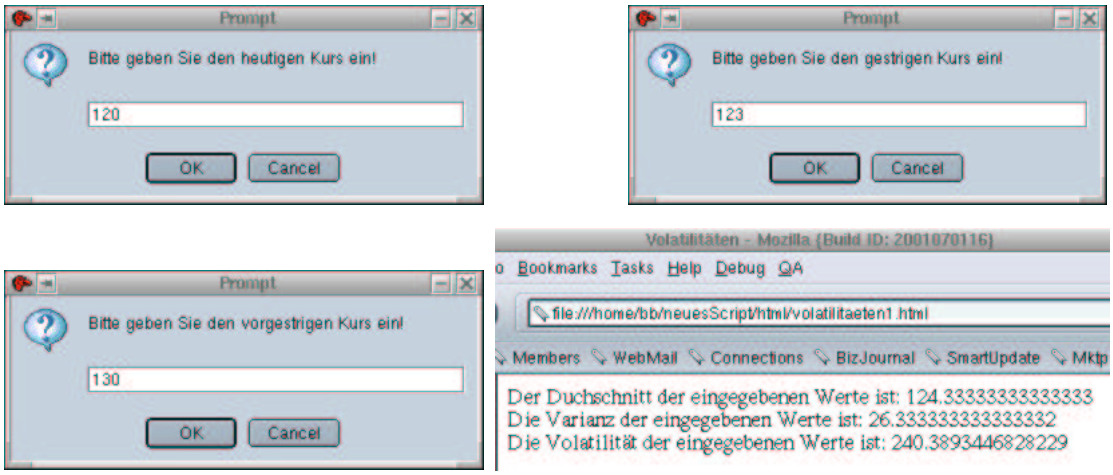


Abbildung 5.3: Volatilitäten Teil 1

Beispiel 5.5 *Volatilitäten Teil 1 in php (html-Datei)*

```
<!-- Volatilitaeten
    Dies ist die Eingabemaske
  Dateiname: volatilitaeten1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Volatilit&auml;ten</title>
</head>
```

```
<body>
<h2>
    Volatilit&auml;ten
</h2>
<form name="volatil1" action="./volatilitaeten1.php" method="post">
    <table border>
        <tr>
            <td>
                Heutiger Kurs
            </td>
            <td>
                <input type="text" name="kursHeute" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Gestriger Kurs
            </td>
            <td>
                <input type="text" name="kursGestern" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Vorgestriger Kurs
            </td>
            <td>
                <input type="text" name="kursVorgestern" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Beispiel 5.6 *Volatilitäten Teil 1 in php (php-Datei)*

```
<!-- Volatilitaeten
    Die Logik
  Dateiname: volatilitaeten1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Volatilit&auml;ten</title>
</head>
<body>
<h2>
    Volatilit&auml;ten: Das Ergebnis
</h2>
```

```
<?php
        // Durchschnittsberechnung
        $durchschnitt=($kursHeute+$kursGestern+$kursVorgestern)/3;
        // Varianz
        $varianz=(($kursHeute-$durchschnitt)*($kursHeute-$durchschnitt) +
                        ($kursGestern-$durchschnitt)*($kursGestern-$durchschnitt) +
                        ($kursVorgestern-$durchschnitt)*($kursVorgestern-$durchschnit
        $dt=3/250;
        //volatilitaet
        $volatilitaet=$varianz/sqrt($dt);
        echo("Der Duchschnitt der eingegebenen Werte ist: " .
                              "$durchschnitt <br>" .
                              "Die Varianz der eingegebenen Werte ist: " .
                              "$varianz <br>" .
                              "Die Volatilit&auml;t der eingegebenen Werte ist: " .
                              "$volatilitaet <br>");
?>
</body>
</html>
```

Abbildung 5.4: Das Volatilitäten-Beispiel in php Teil 1

5.3 Das Raketenbeispiel

In diesem Kapitel wollen wir ein etwas komplexeres Beispiel behandeln. Um die Problemstellung lösen zu können, müssen wir uns kurz mit dem Modulo-Operator und der Integerdivision befassen.

Exkurs: Integerdivision und Modulo-Bildung

Integerdivision und Modulo-Bildung haben an sich wenig mit Informatik zu tun. Es handelt sich hier eher um elementare Mathematik. Zum besseren Verständnis des Folgenden werde ich die Grundgedanken kurz erläutern.

Wenn wir zwei Zahlen durcheinander teilen, ist das Ergebnis entweder eine ganze Zahl (so z.B. wenn wir $6/3$ rechnen, kommt 2 ‘raus) oder ein Bruch. So ist $10/4 = 2,25$.

Es gibt aber Situationen, in denen wir an den Nachkommastellen nicht interessiert sind (wie das folgende Beispiel zeigen wird) und eigentlich nur den ganzzahligen Anteil der Lösung nutzen wollen. Dies nennt man Integerdivision. So ist 10 integerdividiert durch 4 gleich 2. Oder umgangssprachlich formuliert: Integerdivision beantwortet die Frage: Wie oft geht der Nenner in den Zähler. Und in obigem Beispiel: Wie oft geht die 4 in die 10. Übrigens ist auch 11 integerdividiert durch 4 gleich 2. Wie alle sich an die C-Syntax anlehnenden Sprachen bietet weder JavaScript noch php hierfür einen eigenen Operator[4].

Wenn wir in JavaScript oder php eine Integerdivision durchführen wollen, so müssen wir normal dividieren und dann den Nachkomma-Anteil abschneiden. Dafür gibt es Funktionen[5]. In JavaScript heißt die Funktion `Math.floor()`, in php `floor()`.

Wenn man eine Integerdivision durchführt, bleibt ein Rest (der Nachkommaanteil der „normalen“ Division). Diesen erhält man, indem man das Ergebnis der Integerdivision mit dem Nenner multipliziert und das Ergebnis der Multiplikation vom Zähler abzieht. In unserem ersten Beispiel ist der Rest 0. Wir können das einfach ausrechnen, denn der Nenner ist hier 3, der Zähler 6 und das Ergebnis der Integerdivision ist 2. Mit obiger Formel erhalten wir also:

```
Rest von 6 integerdividiert durch 3 =
  6-3* (6 integerdividiert durch 3) = 6-3*2 = 6-6 = 0
```

Im zweiten Beispiel ist der Rest 2:

```
Rest von 10 integerdividiert durch 4 =
  10-4*(10 integerdividiert durch 4) = 10-4*2 = 10-8 = 2
```

Im dritten Beispiel ist der Rest 3:

```
Rest von 11 integerdividiert durch 4 =
  11-4*(11 integerdividiert durch 4) = 11-4*2 = 11-8 = 3
```

Dies nennt man in der Mathematik Modulo-Bildung und sowohl JavaScript, als auch php verfügen dafür über den Operator %. So ist also

```
6%3  = 0,
10%4 = 2 und
11%4 = 3.
```

Ein Beispiel, wo man das anwenden kann (dies ist auch unser Beispiel), ist, wenn wir wissen, wie lange ein Vorgang in Sekunden dauert (z.B. 128 Sekunden), wir aber wissen wollen, wieviele Minuten und Sekunden das sind. Die Beantwortung dieser Frage ist genau Integerdivision und Modulo-Bildung. Denn die Minuten sind die Beantwortung der Frage: Wie oft geht die 60 in die uns gegebenen Sekunden (z.B. wie oft geht die 60 in die 128) und die Sekunden sind der Rest, der übrigbleibt, wenn ich das gemacht habe. Also 128 integerdividiert durch 60 = 2 und 128 modulo 60 = 8, daher sind 128 Sekunden 2 Minuten 8 Sekunden.

Das Raketenbeispiel

Ein Programm soll 3 Werte einlesen, die den Startzeitpunkt einer Rakete in Stunden, Minuten und Sekunden angeben. Danach soll die Flugzeit (in Sekunden) eingelesen werden. Dann soll aus diesen Angaben die Ankunftszeit der Rakete berechnet werden und in einem Fenster ausgegeben werden.

[4] Im Gegensatz zu einigen anderen Programmiersprachen, wie Visual Basic oder Pascal.

[5] Und Sie wissen immer noch nicht genau, was das ist, aber Arbeiten damit geht ja schon.

Wir machen zur Zeit noch eine Einschränkung: Start und Landung der Rakete finden am gleichen Tag statt.

Zunächst überlegen wir uns, was unser Programm ganz grob tun muss. Die Antwort ist: Es muss die Eingaben einlesen, die Eingaben in "Floats" umwandeln (nur JavaScript), dann die Ankunftszeit berechnen und zum Schluss die berechnete Ankunftszeit wieder ausgeben. Aufgeschrieben sieht das so aus:

- Benutzereingaben einlesen
- Eingaben in Floats umwandeln
- Ankunftszeit berechnen
- Ankunftszeit ausgeben

Wir sind jetzt schon einen ganzen Schritt weiter. "Benutzereingaben einlesen" ist einfach, das regeln wir mit prompt-Fenstern oder über ein Formular. "Ankunftszeit ausgeben" realisieren wir, indem wir mit `document.write` bzw. `echo` in die html-Datei schreiben.

"Ankunftszeit berechnen" ist nicht so einfach. Dies können wir zumindestens nicht 1 zu 1 in ein Programm umsetzen. Nichtsdestotrotz müssen wir das Teilproblem lösen.

Und wenn wir nicht so recht weiterwissen, dann betrachten wir –und das machen wir immer so– ein Beispiel.

Also nehmen wir an, unser Benutzer hat 13:56:12 Uhr als Abflugzeit und 4365 Sekun- den als Flugzeit eingegeben. Wir haben in beiden Zeiten Sekunden, die 12 Sekunden der Abflugzeit und die 4365 Sekunden der Flugzeit. Wir könnten also diese beiden Zeiten addieren und erhalten 4377 Sekunden. Wir können jetzt mit Fug und Recht sagen:

Die Rakete landet um 13:56:4377.

Dies ist aber noch nicht recht befriedigend, weil keiner außer uns Zeiten so angibt. Wir müssen also entweder einen neuen Standard für Zeitformate setzen oder weitermachen. Wir machen weiter[6]. Der naheliegende Schritt ist, sich zu überlegen: Wieviele Minuten sind wohl in den 4377 Sekunden?

Dies können wir aber leicht ausrechnen, denn wie wir wissen ist das die Integerdivision

```
4377 integerdividiert durch 60.
```

Dies ist 72. In unseren 4377 Sekunden sind also 72 Minuten. Als nächstes berechnen wir, wieviel Sekunden da wohl übrigbleiben. Das ist das Ergebnis der Modulobildung, also

```
übrigbleibende Sekunden =
  4377-60*(4377 integerdividiert durch 60) = 4377-60*72 =
  4377-4320 = 57
```

Unsere komischen 4377 Sekunden in der Ankunftszeit sind also in Wahrheit 72 Minuten und 57 Sekunden. Beachten Sie, dass wir die „wahren" Sekunden der Ankunftszeit jetzt bereits kennen, unsere Rakete kommt um 57 Sekunden an. Fragt sich nur, zu welcher Stunde und Minute. Wir haben aber die Minuten in den 4377 Sekunden der Ankunftszeit bereits ausgerechnet (waren 72) und wir haben ja bereits Minuten (56) in der Ankunftszeit. Wir könnten also wie oben vorgehen und das einfach mal addieren. Das tun wir und erhalten 128 Minuten. Also behaupten wir jetzt:

Die Rakete landet um 13:128:57.

Leicht besser, aber immer noch nicht gut genug. Aber jetzt gehen wir analog zu unserer Sekunden-Problemlösung vor. Wir überlegen uns einfach: Wieviele Stunden sind in den Minuten?

Dies können wir aber leicht ausrechnen, denn wie wir wissen ist das die Integerdivision

[6] Microsoft würde einen neuen Standard setzen :-).

```
128 integerdividiert durch 60.
```

Dies ist aber 2. In unseren 128 Minuten sind also 2 Stunden. Als nächstes berechnen wir, wieviel Minuten da wohl übrigbleiben, das ist das Ergebnis der Modulobildung, also

```
übrigbleibende Minuten = 128-60*(128 integerdividiert durch 60) =
  128-60*2 = 128-120 = 8
```

Unsere komischen 128 Minuten in der Ankunftszeit sind also in Wahrheit 2 Stunden und 8 Minuten. Beachten Sie, dass wir die „wahren“ Minuten der Ankunftszeit jetzt bereits kennen, unsere Rakete kommt um 8 Minuten an. Da wir die Sekunden der Ankunftszeit auch schon kennen, können wir sogar sagen: Die Rakete kommt um 8 Minuten 57 Sekunden an. Fragt sich nur, zu welcher Stunde. Wir haben aber die Stunden in den 128 Minuten der Ankunftszeit bereits ausgerechnet (waren 2) und wir haben ja bereits Stunden (13) in der Ankunftszeit. Wir könnten also wie oben vorgehen und einfach mal addieren. Das tun wir und erhalten 15 Stunden. Also behaupten wir jetzt:

Die Rakete landet um 15:08:57.

Und das ist perfekt. Wir haben also unser Problem an einem Beispiel gelöst. Nun generalisieren wir das Ganze. Dies entsteht einfach dadurch, dass wir die einzelnen in unserem Zahlenbeispiel ausgeführten Arbeitsschritte aufschreiben.

Zunächst haben wir die Flugzeit in Sekunden zu den Sekunden der Abflugzeit addiert und behauptet (was ja auch stimmt), dass dies die Ankunftszeit ist. Also:

- addiere die Flugzeit zu den Sekunden der Abflugzeit.

Dann haben wir die Minuten in den entstehenden Sekunden ermittelt (durch Integerdivision) und das Ergebnis den Minuten der Abflugzeit zuaddiert. Also:

- addiere die Minuten in den errechneten Sekunden (Integerdivision der errechneten Sekunden durch 60) zu den Minuten der Abflugzeit

Danach haben wir die übrigbleibenden Sekunden ermittelt (durch Modulobildung) und hatten so die Sekunden des Landezeitpunkts.

- weise den Rest obiger Integerdivision (Modulo-Bildung) den auszugebenden Sekunden zu

Daraufhin haben wir die Stunden in den entstehenden Minuten ermittelt (durch Integerdivision) und das Ergebnis den Minuten der Abflugzeit zuaddiert. Also:

- addiere die Stunden in den errechneten Minuten (Integerdivision der errechneten Minuten durch 60) zu den Stunden der Abflugzeit

Im weiteren Verlauf haben wir die übrigbleibenden Minuten ermittelt (durch Modulobildung) und hatten so die Minuten des Landezeitpunkts.

- weise den Rest obiger Integerdivision (Modulo-Bildung) den auszugebenden Minuten zu

Das einzige Problem der obigen Problemlösung entsteht, wenn die Stunden durch die Addition der Stunden in den Minuten größer als 24 werden. Das kann aber nicht passieren, da in der Problemstellung steht, dass die Rakete am gleichen Tag startet und landet.

Nun bilden wir die einzelnen oben aufgeführten Schritte auf Anweisungen in unseren Programmiersprachen JavaScript oder php ab.

- Addiere die Flugzeit zu den Sekunden der Abflugzeit.

JavaScript: `sekunden=sekunden+flugzeit;`
php: `$sekunden=$sekunden+$flugzeit;`

- Addiere die Minuten in den errechneten Sekunden (Integerdivision der errechneten Sekunden durch 60) zu den Minuten der Abflugzeit.
  JavaScript: `minuten=minuten+Math.floor(sekunden/60);`
  php: `$minuten=$minuten+floor($sekunden/60);`

- Weise den Rest obiger Integerdivision (Modulo-Bildung) den auszugebenden Sekunden zu
  JavaScript: `sekunden=sekunden%60;`
  php: `$sekunden=$sekunden%60;`

- Addiere die Stunden in den errechneten Minuten (Integerdivision der errechneten Minuten durch 60) zu den Stunden der Abflugzeit
  JavaScript: `stunden=stunden+Math.floor(minuten/60);`
  php: `$stunden=$stunden+floor($minuten/60);`

- Weise den Rest obiger Integerdivision (Modulo-Bildung) den auszugebenden Minuten zu
  JavaScript: `minuten=minuten%60;`
  php: `$minuten=$minuten%60;`

Nachdem wir so weit gediehen sind, ist die letztendliche Implemetierung der Lösung des Raketenbeispiels einfach Beispiel 5.7 zeigt die Realisierung in JavaScript, Beispiele 5.8 und 5.9 die php-Implementierung. Abb. 5.5 gibt einen beispielhaften Durchlauf durch Beispiel 5.7 wieder, Abb. 5.6 zeigt selbiges für php[7].

Beispiel 5.7 *Das Raketenbeispiel Teil 1*

```
<!-- Raketenbeispiel 1
  Dateiname: raketen1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Abflugzeit einer Rakete <br>
    und sodann die Flugzeit (in Sekunden) ein.<br>
    Die Ankunftszeit wird berechnet.<br>
    <script language = "JavaScript">
    var sekunden;
    var minuten;
    var stunden;
    var flugzeit;
    //Einlesen
    stunden=prompt("Bitte geben Sie die Stunden ein","");
    minuten=prompt("Bitte geben Sie die Minuten ein","");
    sekunden=prompt("Bitte geben Sie die Sekunden ein","");
```

[7] Glücklicherweise sind bei gleichen Eingaben auch die Ergebnisse gleich!

```
    flugzeit=prompt("Bitte geben Sie die Flugzeit ein","");
    //Umwandeln
    stunden=parseInt(stunden);
    minuten=parseInt(minuten);
    sekunden=parseInt(sekunden);
    flugzeit=parseInt(flugzeit);
    //Berechnen, zuerst Sekunden und Minuten
    sekunden=sekunden+flugzeit;
    minuten=minuten+Math.floor(sekunden/60);
    sekunden=sekunden%60;
    //nun minuten und stunden
    stunden=stunden+Math.floor(minuten/60);
    minuten=minuten%60;
    //ausgeben
    document.write("Die Ankunftszeit ist: <br>" +
                   stunden + ":" + minuten + ":" +
                   sekunden);
    </script>
</body>
</html>
```

Abbildung 5.5: Raketenbeispiel Teil 1 in JavaScript

Beispiel 5.8 *Das Raketenbeispiel Teil 1 in php (html-Datei)*

```
<!-- raketenbeispiel 1
```

```
    Dies ist die Eingabemaske
  Dateiname: raketen1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel Teil 1</title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Abflugzeit einer Rakete <br>
    und sodann die Flugzeit (in Sekunden) ein.<br>
    Die Ankunftszeit wird berechnet.<br> <hr>
<form name="raketen1" action="./raketen1.php" method="post">
    <table border>
        <tr>
            <td>
                Abflugzeit Stunden
            </td>
            <td>
                <input type="text" name="stunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Abflugzeit Minuten
            </td>
            <td>
                <input type="text" name="minuten" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Abflugzeit Sekunden
            </td>
            <td>
                <input type="text" name="sekunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Flugzeit
            </td>
            <td>
                <input type="text" name="flugzeit" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
```

```
</html>
```

Beispiel 5.9 *Das Raketenbeispiel Teil 1 in php (php-Datei)*

```
<!-- raketenbeispiel 1
    Die Logik
  Dateiname: raketen1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel Teil 1</title>
</head>
<body>
<h2>
    Raketenbeispiel Teil 1: Das Ergebnis
</h2>
<?php
    //Berechnen, zuerst Sekunden und Minuten
    $sekunden=$sekunden+$flugzeit;
    $minuten=$minuten+floor($sekunden/60);
    $sekunden=$sekunden%60;
    //nun minuten und stunden
    $stunden=$stunden+floor($minuten/60);
    $minuten=$minuten%60;
    //ausgeben
    echo("Die Ankunftszeit ist: <br>" .
                    "$stunden:$minuten:$sekunden");
?>
</body>
</html>
```

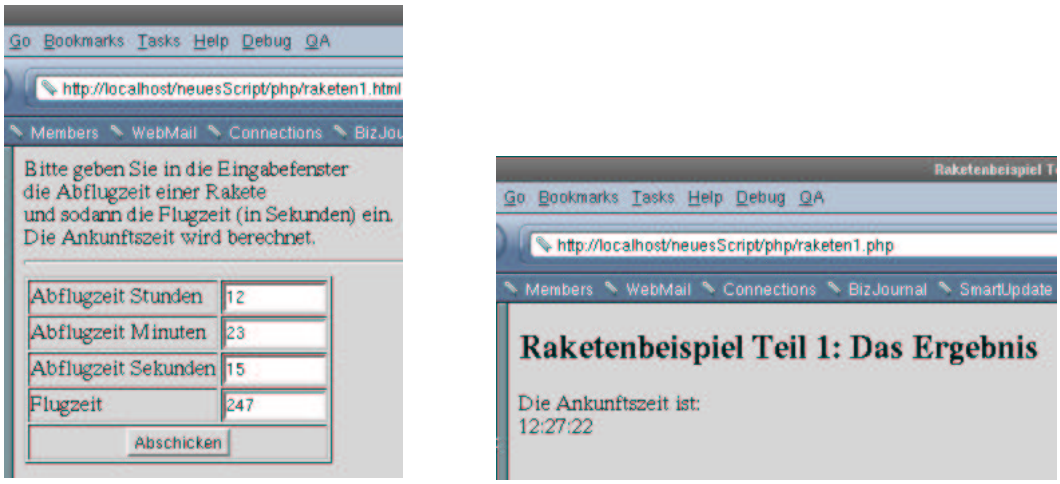


Abbildung 5.6: Das Raketenbeispiel in php Teil 1

Aufgabe 5.1 *Sie sollen für eine Bank die Errechnung von Darlehenskonditionen für Kunden der Bank über das Internet ermöglichen. Eingegeben werden soll das Eigenkapital und der Preis der Immobilie, die gekauft werden soll. Der Zinssatz ist 5 %, die Tilgung 1 %. Das Programm soll die monatliche Belastung ausgeben.*

Aufgabe 5.2 *Das Raketenbeispiel hat keine Gnade vor den Augen Ihrer Anwender gefunden. Die möchten nämlich auch die Flugzeit in Stunden, Minuten und Sekunden eingeben. Ändern Sie die Lösung von Beispiel 5.7, bzw. von Beispielen 5.8 und 5.9.*

Aufgabe 5.3 *Eine weitere Variante des Raketenprogramms soll programmiert werden. Nun möchten Ihre Anwender die Start- und Landezeit einer Rakete eingeben und dann die Flugzeit ausgegeben erhalten! Auch hier gilt allerdings die Einschränkung, dass die Rakete am gleichen Tag startet und landet.*

Kapitel 6

Algorithmen und Kontrollstrukturen (Steuerungsstrukturen)

6.1 Algorithmen

Die Lösung eine Problems zu finden, ist nicht immer wirklich einfach. Dies haben Kapitel 5.3 und auch Aufgabe 5.3 eindeutig gezeigt. Obwohl wir im wesentlichen nicht mehr Sprachelemente der hier betrachteten Programmiersprachen eingesetzt haben, als in allen vorher betrachteten Beispielen, war der Denkaufwand doch wesentlich höher.

Bei den anderen Beispielen konnten wir uns die Aufgabenstellung ansehen und sie dann, ohne groß darüber nachzudenken[1], sofort in ein Programm umsetzen. Bei der Aufgabenstellung in 5.3 war das ganz anders: Zunächst mussten wir die Aufgabenstellung genau verstehen. Und dann brauchten wir einen genauen Lösungsplan. Nichtwissenschaftler[2] würden sagen, wir haben uns „eine Zeichnung gemacht".

Doch schauen wir uns den schon in Kapitel 5.3 dargestellten Ablauf zur Problemlösung noch einmal an:

1. Addieren der Flugzeit in Sekunden zu den Sekunden der Abflugzeit.
2. Addieren der Minuten in den errechneten Sekunden (Integerdivision der errechneten Sekunden durch 60) zu den Minuten der Abflugzeit.
3. Zuweisung des Rests obiger Integerdivision (Modulo-Bildung) zu den auszugebenden Sekunden.
4. Addieren der Stunden in den errechneten Minuten (Integerdivision der errechneten Minuten durch 60) zu den Stunden der Abflugzeit.
5. Zuweisung des Rests obiger Integerdivision (Modulo-Bildung) zu den auszugebenden Minuten.

Folgendes fällt auf:

- Der dargestellte Ablauf ist völlig unabhängig von der Programmiersprache. Wir haben die Problemlösung in JavaScript und php implementiert. Aber wir hätten jede Programmiersprache

[1] ich meine, jeder weiss, wie man zwei Zahlen addiert und jeder kann auch (oder sollte können :-)) genug Prozentrechnung, um bei kredithöhenunabhägigen, konstanten Zinssätzen Belastungen auszurechnen.

[2] Schönes Wort...

wählen können, die modulo-Bildung und Integerdivision direkt unterstützt (und das tun alle[3]). Als wir uns die Lösung überlegt haben, mussten wir also nicht wirklich wissen, wie man z.B. in php modulo-Rechnung programmiert, wichtig für den Ablauf ist nur, dass es geht.

- Der zeitliche Ablauf ist wichtig. Wenn wir Schritt 2 vor Schritt 1 durchführen, werden wir ein nicht unbedingt sinnvolles Ergebnis erhalten. Auch Schritt 2 und 3 zu vertauschen, ist nicht besonders erfolgversprechend[4].

Wir haben also in unserem Ablauf

- die auszuführenden Aktionen und
- die Reihenfolge, in der die Aktionen auszuführen sind,

aufgeschrieben. Und so etwas nennt man einen Algorithmus entwickeln. Ohne dass es uns bewusst war, haben wir so komplizierte Dinge gemacht, wie einen Algorithmus aufzustellen!

Den Algorithmus haben wir dann durch Pseudocode dokumentiert. Das war Ihnen, bevor ich das hier schreibe, sicher auch noch nicht klar, aber genau das haben wir gemacht. Denn Pseudocode ist eine Abfolge einfacher, nicht einmal unbedingt vollständiger Sätze, die den Algorithmus beschreiben. Und wenn wir den von uns entwickelten Ablauf betrachten, genau so sieht er aus!

Pseudocode kann durch reservierte Worte, die Kontrollstrukturen repräsentieren, strukturiert werden. Dies klingt furchtbar kompliziert[5], ist es aber eigentlich nur, weil Sie nicht wissen, was Kontrollstrukturen und reservierte Worte sind. Aber genau das werden Sie im weiteren Verlauf dieses Kapitels erfahren.

Fassen wir das in diesem Kapitel Gesagte noch einmal zusammen:

- Bei einfachen Problemen können wir die Aufgabenstellung nehmen und sie sofort in Programmcode umsetzen.
- Bei komplexeren Aufgabenstellung machen wir uns zunächst einen Plan. Der Plan heißt Algorithmus und wird aufgeschrieben. Die Form, in der wir den Algorithmus aufschreiben, heißt Pseudocode.
- Den Pseudocode können wir dann „problemlos“ in beliebige Programmiersprachen umsetzen.

Die hier vorgestellte Vorgehensweise heißt strukturierte Entwicklung.

6.2 Die if- Anweisung (Ein- und Zweiseitige Auswahl)

6.2.1 Motivation

Unsere bisherigen Programmierkenntnisse lassen nur die Realisierung linearer Programmverläufe zu. Dies bedeutet, die Anweisungen in unseren Programmen werden von oben nach unten in genau vorgegebener Reihenfolge abgearbeitet. Wir können keine Anweisungen nur unter bestimmten Bedingungen durchführen lassen oder Programmblöcke häufiger ablaufen lassen. Abb. 6.1 zeigt die Problematik.

Dies ist aber ein echter Nachteil, wie die nachfolgenden Beispiele veranschaulichen:

[3] Naja, vielleicht bis auf Assembler, aber das kann ich nicht.

[4] Warum?

[5] Ich meine, ich muss ja auch mal schwer verständliche Sätze mit vielen Fremdworten schreiben :-).

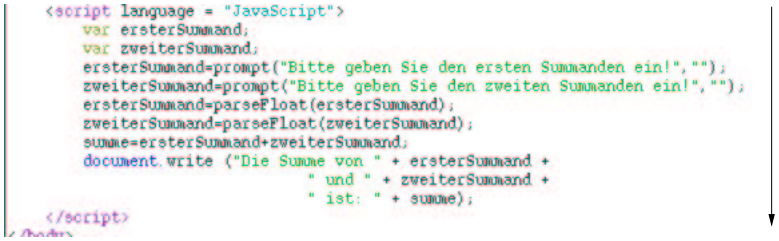


Abbildung 6.1: Linearer Programmablauf

- Wir haben Programme geschrieben, die addieren, multiplizieren und subtrahieren können. Aber für jede Rechenart haben wir ein eigenes Programm, das ist schlecht. Viel schöner wäre es, wenn diese Dinge in einem Programm realisiert wären und der Benutzer entscheiden könnte, was er tun wollte.
- Programmieren wir ein Divisionsprogramm (wie unser Additionsprogramm), können wir Laufzeitfehler nicht vermeiden, denn wenn der Benutzer 0 als Nenner angibt, kann das Programm nicht mehr vernünftig reagieren, weil Divisionen durch Null in der Mathematik ja nicht erlaubt sind. Optimal wäre, den Benutzer in einem solchen Fall auf seinen Eingabefehler aufmerksam zu machen. Wir veranschaulichen uns das an folgenden Beispielen:

Beispiel 6.1 *Division ohne Fehlerabfangen in JavaScript*

```
<!-- Das Programm dividiert zwei einzugebende Zahlen
  Dateiname: division1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division </title>
</head>
<body>
    <script language = "JavaScript">
        var zaehler;
        var nenner;
        var quotient
        zaehler=prompt("Bitte geben Sie den Zähler ein!","");
        nenner=prompt("Bitte geben Sie den Nenner ein!","");
        zaehler=parseFloat(zaehler);
        nenner=parseFloat(nenner);
        quotient=zaehler/nenner;
        document.write ("Der Quotient von " + zaehler +
                                " und " + nenner +
                                " ist: " + quotient);
    </script>
</body>
</html>
```

Abb. 6.2 zeigt, dass JavaScript Dividieren durch Null nicht einmal richtig behandelt. Eine Division durch Null ist in der Mathematik[6] nicht definiert und hat daher kein Ergebnis.

[6]Bis auf in der Non-Standard-Analysis, hier ist das Ergebnis tatsächlich unendlich.

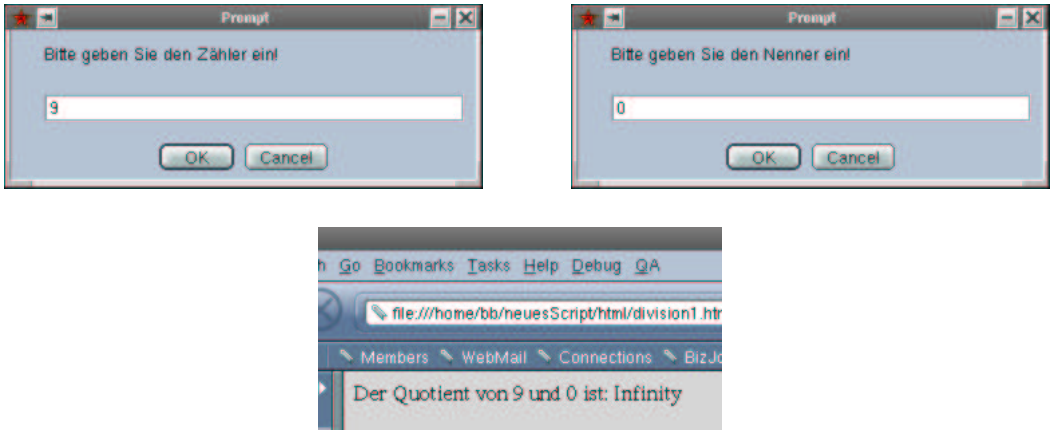


Abbildung 6.2: Division ohne Fehlerabfangen in JavaScript

php verhält sich richtiger (vgl. Abb. 6.3).

Beispiel 6.2 *Division ohne Fehlerabfangen in php, die Eingabeseite*

```
<!-- Das Programm dividiert 2 Zahlen
    Dies ist die Eingabemaske
  Dateiname: division1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division</title>
</head>
<body>
<h2>
    Division zweier Zahlen
</h2>
<form name="division" action="./division1.php" method="post">
    <table border>
        <tr>
            <td>
                Z&auml;hler
            </td>
            <td>
                <input type="text" name="zaehler" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Nenner
            </td>
            <td>
                <input type="text" name="nenner" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
```

```
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Beispiel 6.3 *Division ohne Fehlerabfangen in php, Logik und Ausgabe*

```
<!-- Das Programm dividiert 2 Zahlen
    Die logik
  Dateiname: division1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division (Teil 2)</title>
</head>
<body>
<h2>
    Division zweier Zahlen: Das Ergebnis
</h2>
<?php
    $quotient=$zaehler/$nenner;
    echo ("Der Quotient von $zaehler und $nenner" .
                    " ist: $quotient");
?>
</body>
</html>
```

Abbildung 6.3: Division ohne Fehlerabfangen in php

Wie Abb. 6.3 zeigt, führt php die Division nicht durch. Anstelle dessen wird eine Warnmeldung ausgegeben und die Zeile des Quellcodes, an der der Fehler passierte, referenziert.

6.2.2 Syntax

Lösung solcher Probleme ist die `if`-Anweisung. Sie hat die Form:

```
if (logischer Ausdruck)
{
    anweisung1;
        :
        :
    anweisungN;
}
else
{
    anweisung1nachElse;
        :
        :
    anweisungNnachElse;
}
```

„Logischer Ausdruck“ ist ein Ausdruck, der ein „logisches Ergebnis“ erzeugt.
Ein logisches Ergebnis ist:

```
true    (wahr)
false   (falsch)
```

Logische Ausdrücke sind daher Vergleiche oder bool’sche Ausdrücke (vgl. Kapitel 4.4.3).
Ergibt der logische Ausdruck den Wert true werden die Anweisungen im
`if`-Teil (von nun an `if`-Block genannt),
anderenfalls (der logische Ausdruck ergibt false) werden die Anweisungen im
`else`-Teil (von nun an `else`-Block genannt) ausgeführt.
Regeln:

- Die Zeile mit `if` ... darf nicht mit einem `;` abschließen.
- Vor `else` darf kein `;` stehen.
- `if`-Anweisungen können beliebig tief geschachtelt werden.
- Der `else`-Block ist optional .

Gibt es den `else`-Block nicht, ist dies gleichbedeutend mit: Ansonsten tue nichts. Ist ein `else`-Teil vorhanden, sprechen wir von zweiseitiger Auswahl, ansonsten von einseitiger Auswahl.

`if` und `else` gehören zu den in Kapitel 6.1 erwähnten reservierten Worten. Sie haben für beide Programmiersprachen eine Sonderbedeutung, sie sind ja für die Ablaufsteuerung des Programms zuständig. D.h. sie dürfen vom Entwickler nicht als Variablennamen oder ähnliches verwendet werden.

Wir betrachten nun als Erstes das verbesserte Divisionsprogramm.

Beispiel 6.4 *Division in JavaScript*

```
<!-- Das Programm dividiert zwei einzugebende Zahlen
  Dateiname: division2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division </title>
```

```
</head>
<body>
    <script language = "JavaScript">
        var zaehler;
        var nenner;
        var quotient
        zaehler=prompt("Bitte geben Sie den Zähler ein!","");
        nenner=prompt("Bitte geben Sie den Nenner ein!","");
        if(nenner==0)
        {
            document.write("Versuch durch 0 zu teilen!")
        }
        else
        {
            zaehler=parseFloat(zaehler);
            nenner=parseFloat(nenner);
            quotient=zaehler/nenner;
            document.write ("Der Quotient von " + zaehler +
                                " und " + nenner +
                                " ist: " + quotient);
        }
    </script>
</body>
</html>
```

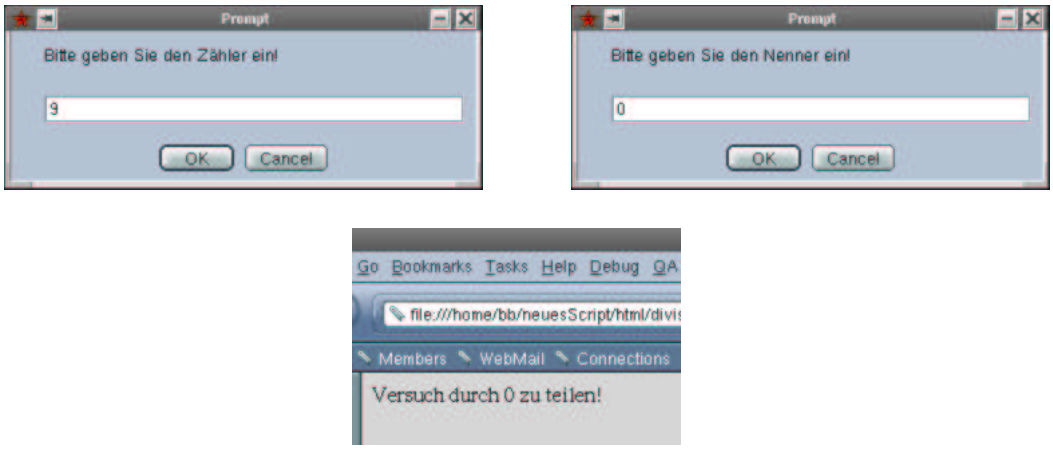


Abbildung 6.4: Division in JavaScript

In Beispiel 6.4 werden wie in Beispiel 6.1 nach der Variablendeklaration `(var zaehler; var nenner; var ergebnis; var eingabe;)` vermittels `prompt(eingabe=prompt(“Zähler:”,“ zaehler= parseFloat(eingabe);eingabe =prompt(“Nenner:”, “”); nenner = parseFloat (eingabe);)` Werte für den Zähler und den Nenner eingelesen und durch `parseFloat` in Zahlen umgewandelt.

Dann wird mittels der `if`-Anweisung getestet, ob der Benutzer als Nenner 0 eingegeben hat `(if (nenner == 0))`. Beachten Sie, dass, wie in Kapitel 4.4 besprochen, Tests auf Gleichheit vermittels == erfolgen. In Abhängigkeit vom Ergebnis des Tests wird entweder “Versuch durch 0 zu teilen!” (der Test hat true ergeben) oder das berechnete Ergebnis (der Test hat false ergeben) durch

document.write in das Browser-Fenster geschrieben. Die Anweisungen im `if`-Block und im `else`-Block der `if`-Anweisung werden in geschweifte Klammern eingeschlossen.

Beachten Sie bitte auch die Einrückungen im dargestellten Programm. Einrücken ist extrem wichtig. Einrücken erhöht die Lesbarkeit der Programme.Komplizierte Entscheidungsstrukturen sind ohne Einrückungen absolut unverständlich.

Nun noch zwei weitere Regeln:

- Wenn im `if`-Block nur eine Anweisung steht, können die geschweiften Klammern weggelassen werden.
- Wenn im `else`-Block nur eine Anweisung steht, können die geschweiften Klammern weggelassen werden.

Dies ist allerdings extrem gefährlich!!!! Man sollte die geschweiften Klammern immer aufnehmen. Grund: Programmänderungen[7].

In php gibt es noch eine weitere Lösung des Divisionsproblems. php verfügt über ein Kommando, um die Programmausführung sofort zu beenden. Dies ist das Kommando `exit()`[8].

Beispiel 6.5 *Division in php, die Eingabeseite*

```
<!-- Das Programm dividiert 2 Zahlen
    Dies ist die Eingabemaske
  Dateiname: division2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division</title>
</head>
<body>
<h2>
    Division zweier Zahlen
</h2>
<form name="division" action="./division2.php" method="post">
    <table border>
        <tr>
            <td>
                Z&auml;hler
            </td>
            <td>
                <input type="text" name="zaehler" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Nenner
            </td>
            <td>
                <input type="text" name="nenner" size=12>
            </td>
        </tr>
        <tr>
```

[7] Überlegen Sie sich selber, warum dies so ist!

[8] Die Klämmerchen sind notwendig.

```
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

Beispiel 6.6 *Division in php, Logik und Ausgabe*

```
<!-- Das Programm dividiert 2 Zahlen
    Die logik
  Dateiname: division2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division (Teil 2)</title>
</head>
<body>
<h2>
    Division zweier Zahlen: Das Ergebnis
</h2>
<?php
    if($nenner==0)
    {
        echo "Versuch durch Null zu teilen!";
        exit();
    }
    $quotient=$zaehler/$nenner;
    echo ("Der Quotient von $zaehler und $nenner" .
                    " ist: $quotient");
?>
</body>
</html>
```

Abb. 6.5 zeigt, dass die php-Lösung ebenfalls das richtige Ergebnis liefert. Gibt der Benutzer Null in das Eingabefeld für den Nenner ein, ist der Wert der php-Variable `$nenner` ebenfalls Null. Da der Test

```
    if($nenner==0)
```

`true`[9] ergibt, wird der `if`-Teil durchgeführt. Dort wird zunächst die Fehlermeldung ausgegeben:

```
        echo "Versuch durch Null zu teilen!";
```

Dann wird die Anweisung `exit()` erreicht. Dadurch wird das Programm abgebrochen. Die Anweisungen

```
    $quotient=$zaehler/$nenner;
    echo ("Der Quotient von $zaehler und $nenner" .
                    " ist: $quotient");
```

[9] Die deutsche Übersetzung ist: wahr :-).

Abbildung 6.5: Division in php

werden nicht erreicht, da das Programm sich ja bereits beendet hat.

Gibt der Benutzer hingegen eine von Null verschiedene Zahl in das Eingabefeld für den Nenner ein, ist der Wert der php-Variable `$nenner` natürlich ebenfalls nicht Null. Da der Test

```
if($nenner==0)
```

nun `false` ergibt, wird der `if`-Teil nicht durchgeführt. Da `exit()` im `if`-Teil steht, wird dies natürlich auch nicht ausgeführt. Einen `else`-Teil gibt es nicht, das heißt, das gesamte `if`-Konstrukt wird ignoriert und die Programmausführung fährt mit der auf das `if`-Konstrukt folgenden Programmzeile fort. Dies ist aber:

```
$quotient=$zaehler/$nenner;
```

Natürlich funktioniert die Lösung mit einem `else`-Teil ebenfalls.

JavaScript verfügt über kein exit()-Kommando.

6.2.3 php und Get und Post

Durch das `if`-Kommando können wir unsere php-Programme vereinfachen. Bislang haben wir, wenn wir in php Benutzereingaben lesen wollten, immer zwei Dateien benötigt: Eine mit einem Formular, in das wir unsere Daten eingeben konnten, und eine, die dann die Aktionen durchgeführt hat. Dies können wir nun ändern.

Wenn ein Browser eine Seite von einem Server anfordert, so wird die Kommunikation zwischen Browser und Client, wie wir bereits wissen, über das http-Protokoll abgewickelt. Als Teil der Anfrage und damit auch Teil des Protokolls überträgt der Browser eine „Methode“. Hier wird im Wesentlichen nur festgelegt, wie dem Server eventuelle Inhalte von Formularfeldern übermittelt werden. Bei der html-Programmierung von Formularen legen wir das über das `method`-Attribut des `form`-Tags fest,

welche Methode beim Abschicken des Formulars genutzt werden soll. Hier gibt es zwei Möglichkeiten: `get` und `post`.

Bei einem „normalen“-Seitenaufruf (bedeutet, Sie clicken auf einen Link oder geben eine neue URL ein) ist die an den Server übermittelte Methode immer `get`.

Als php-Entwickler verfügen wir nun wieder über die Möglichkeit, festzustellen, welche Methode vom Browser beim Seitenaufruf spezifiziert wurde. Denn neben den Variablen mit den Namen etwaiger `Input`-Felder legt php auch noch Variablen an, die sich auf die php-Seite an sich und auf die http-Kommunikation zwischen Browser und Server beziehen.

Um ein php-Programm in einer Seite zu realisieren benötigen wir zwei solcher Variablen:

- `$REQUEST_METHOD`: Diese Variable enthält die vom Browser spezifizierte Methode, also `get` oder `post`.
- `$PHP_SELF`: Dies ist die php-Datei selber.

Nun schauen wir und die Reimplementierung von Beispiel 6.5 und Beispiel 6.6 an.

Beispiel 6.7 *Division in php, Eingabe und Logik in einer Seite*

```
<!-- Das Programm dividiert 2 Zahlen
  Dateiname: division3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division</title>
</head>
<body>
<h2>
    Division zweier Zahlen
</h2>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren

        echo "<form name='division' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Z&auml;hler
                    </td>
                    <td>
                        <input type="text" name="zaehler" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Nenner
                    </td>
                    <td>
                        <input type="text" name="nenner" size=12>
```

```
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        // Anfrage ueber post, das bedeutet, das Formular wurde abgeschickt
        if($nenner==0)
        {
            echo "Versuch durch Null zu teilen!";
            exit();
        }
        $quotient=$zaehler/$nenner;
        echo ("Der Quotient von $zaehler und $nenner" .
                        " ist: $quotient");
    }
?>
</body>
</html>
```

Neues beginnt in Beispiel 6.7 in der Zeile:

```
if($REQUEST_METHOD!="POST")
```

Hier wird überprüft, mit welcher Methode der Aufruf dieser Seite erfolgte. Beim ersten Mal ist dies `get`. Die Bedingung im `if`-Konstrukt ergibt also true[10]. Dies bedeutet, der `if`-Block wird ausgeführt.

In der ersten Zeile des `if`-Blocks

```
echo "<form name='division' action='$PHP_SELF' method='post'>";
```

beginnt der Aufbau des Formulars. Das Kommando an sich ist ein einfaches `echo`. Dies kennen wir bereits aus mehreren Beispielen. Der einzige Unterschied zu Beispiel 6.5 ist hier die Aktion. Durch `action='$PHP_SELF'` wird der Server angewiesen, die gleiche Seite erneut zu laden, nachdem der Benutzer das Formular abgeschickt hat. Dies geschieht einfach dadurch, dass php das `echo`-Kommando durchführt und dabei, wie wir das ja auch schon kennen, die Variable `$PHP_SELF` durch ihren Wert ersetzt (vgl. Abb. 6.6).

Danach wird der php-Interpreter durch `?>` beendet. Es folgt reines html. Wir erkennen hier, dass man php in einer Datei beliebig oft an- und abschalten kann, sogar innerhalb von geschweiften Klammern. Die html-Zeilen erzeugen den Rest unseres Formulars. Sie entsprechen dem Code in Beispiel 6.5. Die Ausführung von php-Code setzt erst wieder mit den Zeilen

```
<?php
    }
```

[10] Selbsverständlich hätten wir statt ungleich post auch gleich get abfragen können. In fast allen Büchern und Internet-Quellen wird aber auf post abgefragt, sodass ich dies in diesen Unterlagen auch so halte.

Der Quellcode von division3.php, wie ihn der Browser beim ersten Aufruf sieht.

```
<!-- Das Programm dividiert 2 Zahlen
  Dateiname: division3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division</title>
</head>
<body>
<h2>
    Division zweier Zahlen
</h2>
<form name='division' action='/neuesScript/php/division3.php' method='post'>
      <table border>
                <tr>
                    <td>
                        2&auml;hler
                    </td>
                    <td>
                        <input type="text" name="zaehler" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Nenner
                    </td>
                    <td>
                        <input type="text" name="nenner" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
      </table>
</form>
</body>
</html>
```

Der Quellcode von division3.php, wie ihn der Browser beim zweiten Aufruf sieht.

```
<!-- Das Programm dividiert 2 Zahlen
  Dateiname: division3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Division</title>
</head>
<body>
<h2>
    Division zweier Zahlen
</h2>
Der Quotient von 7 und 4 ist: 1.75
</body>
</html>
```

Abbildung 6.6: Die Browser-Sicht auf den Source-Code von Beispiel 6.7

ein. Die geschweifte Klammer schließt den `if`-Block ab. Es folgt der `else`-Block, der jedoch nicht durchgeführt wird, weil die Auswertung der Bedingung im `if`-Konstrukt ja true ergeben hatte. Mit dem Ende des `else`-Blocks endet aber auch der php-Code. Die restlichen html-Zeilen beschließen die html-Seite und der diese Seite anfordernde Browser erhält den in Abb. 6.6 dargestellten html-Source-Code.

Schickt der Benutzer durch Klicken auf den “Abschicken”-Button das Formular ab, so startet der Server dadurch, dass wir ja das `$action`-Attribut des `form`-Tags auf `$PHP_SELF` gesetzt hatten, unser Programm ein zweites Mal.

Da aber bei diesem Aufruf durch `method='post'` in der Zeile

```
echo "<form name='division' action='$PHP_SELF' method='post'>";
```

die Anfragemethode auf `post` gesetzt haben, ist das Ergebnis der Auswertung des Vergleichs

```
if($REQUEST_METHOD!="POST")
```

nun false. Dies bedeute, der `if`-Block wird ignoriert und der `else`-Block durchgeführt. Der php-Code im `else`-Block entspricht Beispiel 6.6. Der Quotient wird berechnet. Das Ergebnis wird an den Browser geschickt (vgl. Abb. 6.6).

6.2.4 Beispiele

Euro-Dollar-Umrechnung (fortgesetzt)

Unser Programm zur Euro-Dollar-Umrechnung soll verbessert werden. Das Programm soll nun nicht nur Euro-Beträge in Dollar umrechnen können, sondern auch Dollar-Beträge in Euro. Dazu müssen die Anwender die Zielwährung eingeben können. Ausgegeben werden soll dann das Ergebnis der Umrechnung.

Als Angabe für die Zielwährung Dollar ist Dollar oder dollar erlaubt für den Euro entsprechend euro oder Euro. Bei anderen Eingaben soll die Anwendung eine Fehlermeldung ausgeben und die erlaubten Eingaben darstellen.

Beispiel 6.8 zeigt eine Lösung dieser Problemstellung in JavaScript.

Beispiel 6.8 *Euro-Dollar-Umrechnung Teil 2*

```
<!-- Euro-Dm Umrechnung Teil 2
  Dateiname: euro2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 2</title>
</head>
<body>
    <script language = "JavaScript">
        var zielwaehrung;
        var eurobetrag;
        var dollarbetrag;
        var kurs=0.9;
        zielwaehrung=prompt("Bitte geben Sie die Zielwährung ein!","");
        // von euro nach dollar?
        if((zielwaehrung=="Dollar")||(zielwaehrung=="dollar"))
        {
            eurobetrag=prompt("Bitte geben Sie den Euro-Betrag ein!","");
            eurobetrag=parseFloat(eurobetrag);
            dollarbetrag=eurobetrag*kurs;
            document.write(eurobetrag + " Euro entsprechen " +
                           dollarbetrag + " Dollar");
        }
        else
        {
            // von dollar nach euro?
            if((zielwaehrung=="euro")||(zielwaehrung=="Euro"))
            {
                dollarbetrag=prompt("Bitte geben Sie den Dollar-Betrag ein!","");
                dollarbetrag=parseFloat(dollarbetrag);
                eurobetrag=dollarbetrag*(1/kurs);
                document.write(dollarbetrag + " Dollar entsprechen " +
                               eurobetrag + " Euro");
            }
            else
            {
                // nicht von euro nach dollar und auch nicht andersrum
                // falsche Zielwaehrung
                document.write("Falsche Zielw&auml;hrung: <br>" +
                               "Erlaubt sind: Euro oder Dollar!");
            }
        }
    </script>
</body>
</html>
```

In der Zeile

```
if((ziehlwaehrung=="Dollar")||(ziehlwaehrung=="dollar"))
```

prüfen wir, ob Dollar die Zielwährung ist (beachten Sie die Klammern und die Nutzung von == als Vergleichsoperator). Ist dies der Fall, verzweigt JavaScript in den `if`-Block. Hier wird die Umrechnung durchgeführt und das Ergebnis ausgegeben. Dies entspricht Beispiel 5.1.

Ist dies nicht der Fall, wird der `else`-Block durchlaufen. Hier wird zunächst geprüft, ob die Zielwährung Euro ist:

```
if((ziehlwaehrung=="euro")||(ziehlwaehrung=="Euro"))
```

Ist die Bedingung erfüllt, wird der `if`-Block dieses Vergleichs durchgeführt. Hier wird die Umrechnung durchgeführt und das Ergebnis ausgegeben. Dies entspricht Beispiel 5.1, mit der Ausnahme, dass wir nicht mit dem Dollarkurs an sich, sondern mit dem reziproken Wert des Dollarkurses multiplizieren.

Ist die Bedingung nicht erfüllt, war die Eingabe weder euro noch Euro. Sie war aber auch nicht Dollar oder dollar, denn sonst hätten wir den `else`-Block des ersten Vergleichs nicht erreicht. Dies bedeutet, dass der Benutzer eine Fehleingabe getätigt hat. Im `else`-Block des zweiten Vergleichs wird also die Fehlermeldung ausgegeben.

Abb. 6.7 zeigt beispielhafte Anwendungen des Programms. Die Abbildung zeigt uns auch noch einen Nachteil der bisherigen Programmierung: Wir können bislang keine Ausgaben mit einer begrenzten Anzahl Nachkommastellen erzwingen!

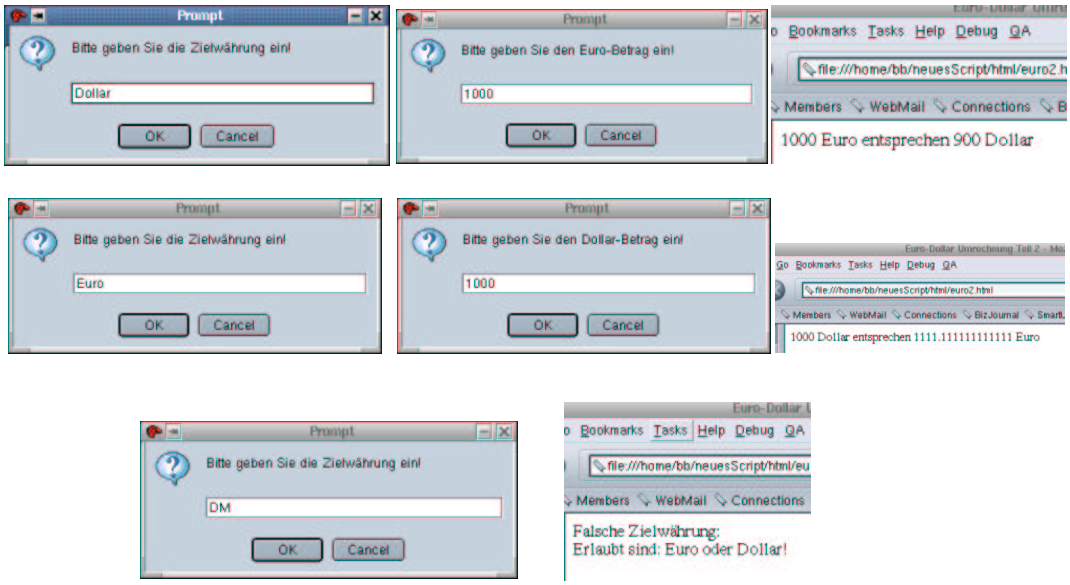


Abbildung 6.7: Das Euro-Dollar Beispiel Teil 2

Die Lösung des Problems in php ist ziemlich ähnlich und in Beispiel 6.9 dargestellt.

Beispiel 6.9 *Euro-Dollar-Umrechnung Teil 2*

```
<!-- Programm zur Euro-Dollar Umrechnung Teil2
     Die Logik
  Dateiname: euro2.php //-->
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Teil 2</title>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='euro2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Zielw&auml;hrung
                    </td>
                    <td>
                        <input type="text" name="zielwaehrung" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Betrag
                    </td>
                    <td>
                        <input type="text" name="betrag" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        $kurs=0.9;
        if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
        {
            $dollarbetrag=$kurs*$betrag;
            echo "$betrag Euro entspricht $dollarbetrag Dollar!";
        }
        else
        {
            if(($zielwaehrung=="euro")||($zielwaehrung=="Euro"))
            {
                $eurobetrag=(1/$kurs)*$betrag;
                echo "$betrag Dollar entsprechen $eurobetrag Euro";
            }
```

```
            else
            {
                echo("Falsche Zielw&auml;hrung: <br>" .
                        "Erlaubt sind: Euro oder Dollar!");
            }
        }
    }
?>
</body>
</html>
```

Da php, wie wir wissen, serverseitig ausgeführt wird und daher keine Eingabefenster kennt, müssen wir die Benutzer über ein Formular eingeben lassen, was dann wieder an den Server geschickt wird. Wir verwenden hier die Vorgehensweise aus Kapitel 6.2.3, damit wir nur eine Datei benötigen. In der Zeile

```
if($REQUEST_METHOD!="POST")
```

wird festgestellt, welche Methode in der Anfrage übermittelt wurde. Ist dies nicht `post`, handelt es sich um den ersten Aufruf der Seite. Das Formular muss also erzeugt werden. Dies geschieht im ersten `if`-Block.

Ist die Methode hingegegen `post`, wird unsere Seite zum zweiten Mal aufgerufen, was bedeutet, dass die Benutzer das Formular bereits gesehen und (hoffentlich[11]) ausgefüllt haben. Wir können also über die Namen der `Input`-Felder der Formulare auf die Eingaben der Benutzer zugreifen und die Berechnung durchführen. Dies ist aber völlig analog zur JavaScript-Lösung.

Abb. 6.8 zeigt beispielhafte Anwendungen des Programms.

Volatilitäten (fortgesetzt)

Volatilität Bewertung kommt noch

Raketenbeispiel (fortgesetzt)

Zum Abschluss dieses Kapitels wollen wir das Raketenbeispiel erweitern. Wir benutzen hier (und im weiteren Verlauf) allerding die in Aufgabe 5.3 modifizierte Version. Wir fügen folgende weitere Abänderungen hinzu:

- Die Rakete startet und landet im gleichen Monat und nicht mehr am gleichen Tag.
- Wenn der Landzeitpunkt vor dem Startzeitpunkt liegt, soll das Programm eine Fehlermeldung ausgeben und abbrechen.
- Wenn die Rakete mehrere Tage unterwegs ist, soll die Anzahl Tage mit ausgegeben werden, erfolgen Start und Landung jedoch am selben Tag, sollen Tage gar nicht auftauchen.

Auch hier müssen wir zur Problemlösung etwas überlegen. Zunächst gewärtigen wir uns noch einmal den Pseudocode des Algorithmus der Lösung zur Berechnung der Flugzeit(vgl. Kapitel 11):

1. Umrechnen der Startzeit in Sekunden

[11] Wir fangen ja noch keinerlei Eingabefehler ab.

Abbildung 6.8: Das Euro-Dollar Beispiel Teil 2 in php

2. Umrechnen der Landezeit in Sekunden
3. Flugzeit in Sekunden = Landezeit in Sekunden - Startzeit in Sekunden
4. vorläufige Minuten der Flugzeit = Flugzeit in Sekunden integerdividiert durch 60
5. Sekunden der Flugzeit = Flugzeit in Sekunden modulo 60
6. Stunden der Flugzeit = vorläufige Minuten der Flugzeit integerdividiert durch 60
7. Minuten der Flugzeit = vorläufige Minuten der Flugzeit modulo 60

Eigentlich können wir diesen Algorithmus wiederverwenden. Es gibt nur zwei kleinere Probleme. Das erste Problem ist: Wie rechnen wir die Start-(oder Landezeit) in Sekunden um? Aber auch mit diesem Problem werden wir fertig. Wir wissen ja, dass ein Tag 24 Stunden hat. Und eine Stunde, das wissen wir auch, hat 3600 Sekunden. Wir müssen ja den Start- und Landetag sowieso einlesen; also multiplizieren wir Start- und Landetag jeweils mit 24*3600 und können den obigen Algorithmus weiterverwenden.

Wenn wir diese Vorgehensweise einmal näher betrachten, sehen wir, dass wir mit unserer neuen Lösung den Bezugspunkt verschoben haben. In der Lösung zu Aufgabe 5.3 haben wir die Anzahl Sekunden betrachtet, die seit Beginn des Tages vergangen sind. In unserer jetzigen Lösung betrachten wir die Anzahl Sekunden die seit Beginn des Monats vergangen sind. Wir haben halt den Bezugspunkt, an dem wir das Zählen der Sekunden starten, nach hinten verschoben.

Das zweite Problem ist analog, wir müssen den Algorithmus zur Rückwandlung der Sekunden der Flugzeit ändern. Denn nun kann die Rakete ja mehrere Tage fliegen[12]. Das bedeutet aber nichts anderes, als dass sich in den Stunden der Flugzeit auch Tage verbergen können. Und hier kommen Neuheiten auf uns zu: Wir müssen jetzt nicht mehr durch 60 integerdividieren bzw. modulo rechnen, sondern, weil ein Tag ja 24 Stunden hat, benutzen wir die 24.

Unter Berücksichtigung der zweiten Neuerung (Startzeit muss vor Landezeit liegen) erhalten wir folgenden neuen Algorithmus:

1. Umrechnen der Startzeit in Sekunden
   (starttag*24*3600+startStunden*3600+startMinuten*60+startSekunden)
2. Umrechnen der Landezeit in Sekunden
   (landetag*24*3600+landeStunden*3600+landeMinuten*60+landeSekunden)
3. Flugzeit in Sekunden = Landezeit in Sekunden - Startzeit in Sekunden
   if(Flugzeit in Sekunden < 0)
   {
   - Fehlermeldung ausgeben

   }
   else
   {
   (a) vorläufige Minuten der Flugzeit = Flugzeit in Sekunden integerdividiert durch 60
   (b) Sekunden der Flugzeit = Flugzeit in Sekunden modulo 60
   (c) vorläufige Stunden der Flugzeit = vorläufige Minuten der Flugzeit integerdividiert durch 60
   (d) Minuten der Flugzeit = vorläufige Minuten der Flugzeit modulo 60
   (e) Tage der Flugzeit = vorläufige Stunden der Flugzeit integerdividiert durch 24
   (f) Stunden der Flugzeit = vorläufige Stunden der Flugzeit modulo 24

   }

An diesem Beispiel sehen wir auch, wie wir Bedingungen in Pseudocode formulieren können. Wir benutzen das `if`-Konstrukt unserer Programmiersprachen. Die Bedingung schreiben wir im Klartext in die runden Klammern. Nun setzen wir den Algorithmus in ein JavaScript- bzw. php-Programm um. Zunächst das JavaScript-Programm:

Beispiel 6.10 *Raketenprogramm Teil 2*

```
<-- Raketenbeispiel 2 des Textes
  Dateiname: raketen2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
```

[12] Was bei einem Flug z.B. zum Mond ja eigentlich auch wahrscheinlich ist.

```
die Startzeit einer Rakete <br>
und sodann die Landezeit ein.<br>
Die Flugzeit wird berechnet.<br>
<script language = "JavaScript">
    var startSekunden;
    var startMinuten;
    var startStunden;
    var starttag
    var startzeitInSekunden;
    var landeStunden;
    var landeMinuten;
    var landeSekunden;
    var landetag;
    var landezeitInSekunden;
    var flugzeitInSekunden;
    var flugzeitStunden;
    var flugzeitMinuten;
    var flugzeitSekunden;
    var flugzeitTage;
    //Einlesen
    starttag=prompt("Bitte geben Sie den Starttag ein","");
    startStunden=prompt("Bitte geben Sie die Stunden der Startzeit ein","");
    startMinuten=prompt("Bitte geben Sie die Minuten der Startzeit ein","");
    startSekunden=prompt("Bitte geben Sie die Sekunden der Startzeit ein","");
    landetag=prompt("Bitte geben Sie den Landetag ein","");
    landeStunden=prompt("Bitte geben Sie die Stunden der " +
                        "Landezeit ein","");
    landeMinuten=prompt("Bitte geben Sie die Minuten der " +
                        "Landezeit ein","");
    landeSekunden=prompt("Bitte geben Sie die Sekunden der " +
                        "Lande ein","");

    //Umwandeln
    starttag=parseInt(starttag);
    startStunden=parseInt(startStunden);
    startMinuten=parseInt(startMinuten);
    startSekunden=parseInt(startSekunden);
    landetag=parseInt(landetag);
    landeStunden=parseInt(landeStunden);
    landeMinuten=parseInt(landeMinuten);
    landeSekunden=parseInt(landeSekunden);
    // start-und landezeit in sekunden umrechnen
    startzeitInSekunden=starttag*24*3600+
            startStunden*3600+startMinuten*60+startSekunden;
    landezeitInSekunden=landetag*24*3600+
            landeStunden*3600+landeMinuten*60+landeSekunden;
    // flugzeitInSekunden berechnen
    flugzeitInSekunden=landezeitInSekunden-startzeitInSekunden;
    if(flugzeitInSekunden<0)
    {
        document.write("Fehleingabe: Landezeit vor Startzeit!");
    }
    else
    {
```

```
            //Flugzeit umrechnen, zuerst Sekunden und Minuten
            flugzeitMinuten=Math.floor(flugzeitInSekunden/60);
            flugzeitSekunden=flugzeitInSekunden%60;
            //nun minuten und stunden
            flugzeitStunden=Math.floor(flugzeitMinuten/60);
            flugzeitMinuten=flugzeitMinuten%60;
            flugzeitTage=Math.floor(flugzeitStunden/24);
            flugzeitStunden=flugzeitStunden%24;
            //ausgeben
            if(flugzeitTage==0)
            {
                document.write("Die Flugzeit betr&auml;gt: <br>" +
                               flugzeitStunden + " Stunden <br>" +
                               flugzeitMinuten + " Minuten <br>" +
                               flugzeitSekunden + " Sekunden <br>");
            }
            else
            {
                document.write("Die Flugzeit betr&auml;gt: <br>" +
                               flugzeitTage + " Tage <br>" +
                               flugzeitStunden + " Stunden <br>" +
                               flugzeitMinuten + " Minuten <br>" +
                               flugzeitSekunden + " Sekunden <br>");
            }
        }
    </script>
</body>
</html>
```

Abb. 6.9 zeigt eine beispielhafte Anwendung des Programms.

Zum Abschluss die Realisierung in php und ein beispielhafter Durchlauf durch das php-Programm.

Beispiel 6.11 *Raketenbeispiel Teil 2 in php*

```
<!-- raketenbeispiel 2 des Textes
  Dateiname: raketen2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br> <hr>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren
```

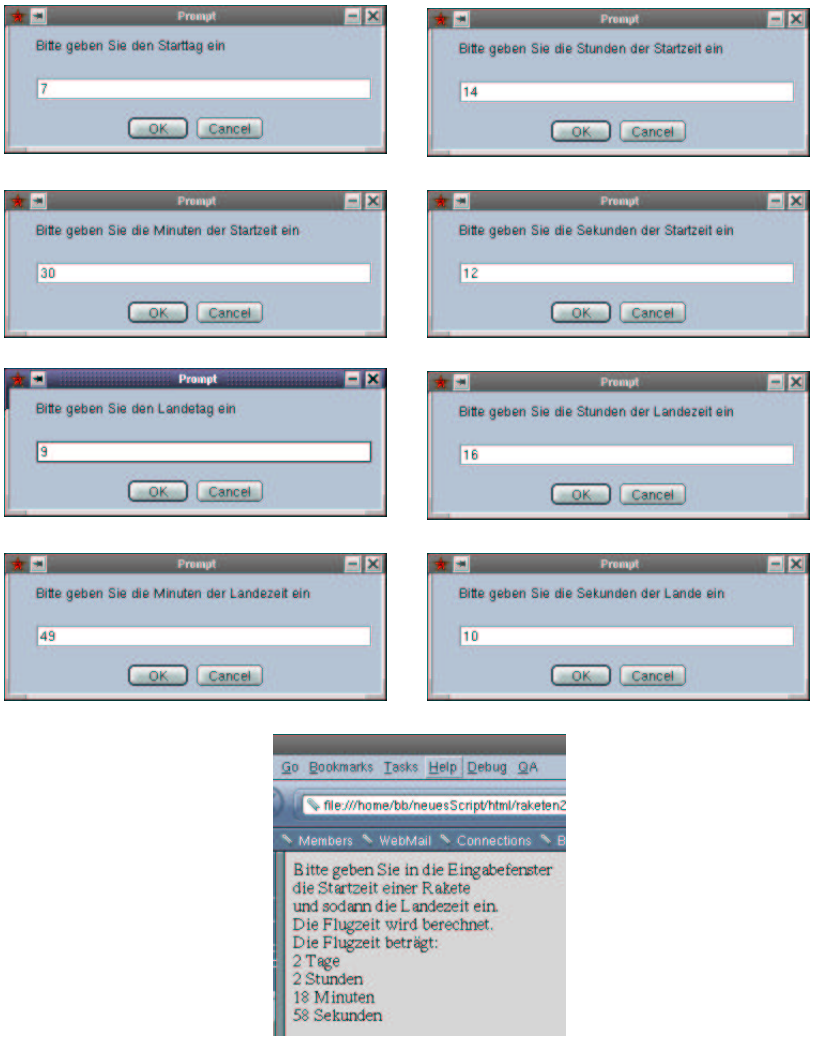


Abbildung 6.9: Das Raketenbeispiel Teil 2

```
        echo "<form name='raketen2' action='$PHP_SELF' method='post'>";
?>
        <table border>
            <tr>
                <td>
                    Starttag
                </td>
                <td>
                    <input type="text" name="starttag" size=12>
                </td>
            </tr>

            <tr>
                <td>
                    Startzeit Stunden
                </td>
```

```
    <td>
        <input type="text" name="startStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Minuten
    </td>
    <td>
        <input type="text" name="startMinuten" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Sekunden
    </td>
    <td>
        <input type="text" name="startSekunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landetag
    </td>
    <td>
        <input type="text" name="landetag" size=12>
    </td>
</tr>

<tr>
    <td>
        Landezeit Stunden
    </td>
    <td>
        <input type="text" name="landeStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landezeit Minuten
    </td>
    <td>
        <input type="text" name="landeMinuten" size=12>
    </td>
</tr>
<tr>
    <td>
        Landezeit Sekunden
    </td>
    <td>
        <input type="text" name="landeSekunden" size=12>
    </td>
</tr>
<tr>
```

```
                <td colspan="2" align="center">
                    <input type="submit" name="Button1" value="Abschicken">
                </td>
            </tr>
        </table>
        </form>
<?php
    }
    else
    {
        // zweiter Aufruf nun rechnen
        $startzeitInSekunden=$starttag*24*3600+
                    $startStunden*3600+$startMinuten*60+$startSekunden;
        $landezeitInSekunden=$landetag*24*3600+
                    $landeStunden*3600+$landeMinuten*60+$landeSekunden;
        // flugzeitInSekunden berechnen
        $flugzeitInSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitInSekunden<0)
        {
            echo "Fehleingabe: Landezeit vor Startzeit!";
        }
        else
        {
            //Flugzeit umrechnen, zuerst Sekunden und Minuten
            $flugzeitMinuten=floor($flugzeitInSekunden/60);
            $flugzeitSekunden=$flugzeitInSekunden%60;
            //nun minuten und stunden
            $flugzeitStunden=floor($flugzeitMinuten/60);
            $flugzeitMinuten=$flugzeitMinuten%60;
            $flugzeitTage=floor($flugzeitStunden/24);
            $flugzeitStunden=$flugzeitStunden%24;
            //ausgeben
            if($flugzeitTage==0)
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
            else
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitTage Tage <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
        }
    }
?>
</body>
</html>
```

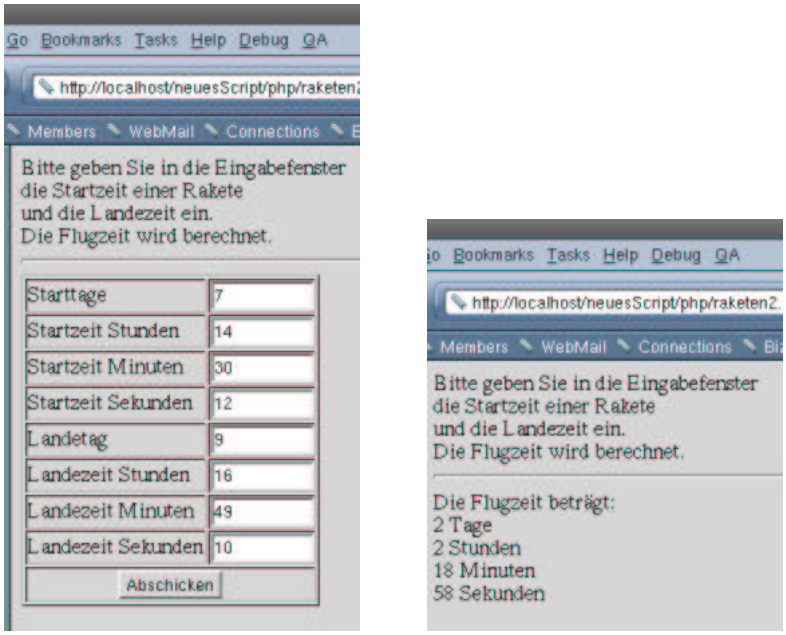


Abbildung 6.10: Das Raketenbeispiel Beispiel Teil 2 in php

Aufgabe 6.1 *Sie sollen für eine Bank die Errechnung von Darlehenskonditionen für Kunden der Bank über das Internet ermöglichen. Eingegeben werden soll das Eigenkapital und der Preis der Immobilie, die gekauft werden soll. Der Zinssatz ist 5 %, die Tilgung 1 %. Das Programm soll die monatliche Belastung ausgeben. Wenn die Eigenkapitalquote des Kunden kleiner als 30 % ist, soll keine Berechnung durchgeführt werden und anstelle dessen ausgegeben werden, dass die Bank Immobilienerwerb mit einer so geringen Eigenkapitalquote nicht finanziert.*

Aufgabe 6.2 *Eine andere Bank vergibt Kredite für das Privatkundengeschäft nach folgenden Kriterien:*

- *Zunächst wird der Preis der Immobilie ermittelt (indem der Kunde ihn angibt).*
- *Dann wird das Eigenkapital ermittelt (indem der Kunde es angibt).*
- *Dann wird die Tilgung ermittelt (indem der Kunde sie angibt).*
- *Bei bereits bestehenden Immobilien entspricht der Wert der Immobilie dem Kaufpreis. Wenn es sich hingegen um einen Neubau handelt zieht die Bank 20% vom Kaufpreis ab, um den Wert der Immobilie zu ermitteln.*
- *Sodann gibt es 4 Zinsbereiche:*
  - *Für das aufzunehmende Geld bis zu 60% des Wertes der Immobile 6,25%.*
  - *Für das aufzunehmende Geld zwischen 60% und 80 % des Wertes der Immobile 7%.*
  - *Für das aufzunehmende Geld zwischen 80% und 100 % des Wertes der Immobile 7,5%.*
  - *Für das aufzunehmende Geld über 100% des Wertes der Immobile 8,5%. (das kann wg. des 20% Abzugs bei Neubauten passieren).*

Diese Bank hat keine Skrupel und verleiht ihr Geld auch, wenn der Kunde kein Eigenkapital hat (Hauptsache die Sicherheit stimmt). Beispiele für die Zinsberechnung:

Der Kunde erwirbt eine bereits bestehende Immobilie für 100000 Euro (kein Neubau) und hat kein Eigenkapital. Demzufolge muss er auch 100000 Euro aufnehmen. Dann zahlt er:

1. *für 60000 Euro Kredit 6,25% Zinsen*
2. *für 20000 Euro Kredit 7% Zinsen (Bereich zwischen 60% und 80%)*
3. *für weitere 20000 Euro Kredit 7,5% Zinsen (Bereich zwischen 80% und 100%)*

Insgesamt zahlt er die Summe der drei obigen Punkte. Ist die Immobilie sogar ein Neubau, zieht die Bank ja 20% vom Immobilienpreis ab. Dies bedeutet der Wert der Immobilie ist nur 80000 Euro. 60% von 80000 Euro sind 48000 Euro. In diesem Fall zahlt der Kunde:

1. *für 48000 Euro Kredit 6,25% Zinsen*
2. *für 16000 Euro Kredit 7% Zinsen (Bereich zwischen 60% und 80%)*
3. *für weitere 16000 Euro Kredit 7,5% Zinsen (Bereich zwischen 80% und 100%)*
4. *und für die letzten 20000 Euro Kredit 8,5% Zinsen (Bereich über 100%)*

Insgesamt zahlt er die Summe der drei obigen Punkte. Natürlich kommt der Tilgungsanteil noch hinzu. Weitere Beispiele überlegen Sie sich selbst.

Auch diese Bank möchte es ihren Kunden ermöglichen, sich die monatliche Belastung über das Internet anzeigen zu lassen. Und Sie sollen das entwickeln.

6.3 Der Switch (Mehrfachauswahl)

6.3.1 Motivation

Stellen wir uns vor, dass wir in eine Internet-Seite Taschenrechnerfunktionalität[13] integrieren wollen. Unser Taschenrechner soll addieren, multiplizieren, subrahieren und dividieren können. Die Benutzer geben die zu bearbeitenden Zahlen und den Operator ein. Bei einem falschen Operator soll eine entsprechende Fehlermeldung ausgegeben werden.

Dies ist ein Beispiel, das jeder von Ihnen mittlerweise sofort implementieren können sollte. Beispiel 6.12 zeigt eine Lösung in php.

Beispiel 6.12 *Taschenrechner in php*

```
<!--  Taschenrechner
  Dateiname: taschenrechner.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Taschenrechner </title>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
```

[13] Dies ist zugegebenermaßen nicht das intelligenteste Beispiel, da Taschenrechner ja teilweise schon mit Kugelschreibern ausgeliefert werden und jedes halbwegs anständige Handy auch einen hat!

```
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='taschenrechner' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Erster Operand
                    </td>
                    <td>
                        <input type="text" name="ersterOperand" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Operator
                    </td>
                    <td>
                        <input type="text" name="operator" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Zweiter Operand
                    </td>
                    <td>
                        <input type="text" name="zweiterOperand" size=12>
                    </td>
                </tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        if($operator=="+")
        {
            $ergebnis=$ersterOperand+$zweiterOperand;
        }
        else
        {
            // war nicht +, vielleicht -
            if($operator=="-")
            {
                $ergebnis=$ersterOperand-$zweiterOperand;
            }
            else
            {
                // auch nicht - vielleicht *
```

```
                if($operator=="*")
                {
                    $ergebnis=$ersterOperand*$zweiterOperand;
                }
                else
                {
                // auch nicht * vielleicht /
                    if($operator=="/")
                    {
                        //Teilen durch 0 verhindern
                        if($zweiterOperand!=0)
                        {
                            $ergebnis=$ersterOperand/$zweiterOperand;
                        }
                        else
                        {
                            $ergebnis="Versuch durch 0 zu teilen!";
                        }
                    }
                    else
                    {
                        // auch nicht /, Fehleingabe
                        $ergebnis="Falscher Operator eingegeben!";
                    } //schliesst else zu if /
                }//schliesst else zu if *
            }//schliesst else zu if -
        }//schliesst else zu if +
        echo "Das Ergebnis ist: $ergebnis";
    }//schliesst else zu if REQUEST_METHOD

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

Die Variable $ergebnis in Beispiel 6.12 enthält bei korrekten Eingaben das berechnete Ergebnis, bei falschen Eingaben die Fehlermeldung. Solche Art der Codierung ist nur möglich, weil Variablen in php (und auch in JavasScript) den Typ wechseln können. Abb. 6.11 zeigt beispielhafte Ein- und Ausgaben.

Nachteil an dieser Lösung ist die Unübersichtlichkeit durch die Verschachtelung der `if`-Anweisungen. Insbesondere das Ende des Programms, wenn die Klammern der ineinander geschachtelten `else`-Blöcke geschlossen werden, ist eine potentielle Fehlerquelle. Der Grund ist der Missbrauch der `if`-Anweisung. `if` sollte eigentlich bei zweiseitigen Entscheidungssituationen eingesetzt werden. Dies ist hier aber nicht der Fall. Wir unterscheiden zwischen fünf Fällen (+,-,*,/, Fehleingabe). Für solche Entscheidungssituationen bieten eigentlich alle Programmiersprachen eigene Konstrukte. In php und JavaScript ist dies der Switch.

6.3.2 Syntax

Lösung solcher Probleme ist die `switch`-Anweisung. Sie hat die Form:

```
switch (selector)
```

Abbildung 6.11: Das Taschenrechnerbeispiel in php

```
{
   case marke1:
            anweisung1.1;
                ...
            anweisung1.N;
            break;
   case marke2 :
            anweisung2.1;
                ...
            anweisung2.N;
            break;
    ...

   case markeN :
            anweisungN.1;
                ...
            anweisungN.N;
            break;
   default :
            anweisungD.1;
                ...
            anweisungD.N;
            break;
}
```

Hierbei ist selector ein Ausdruck, der einen String, eine Zahl oder einen Boolean ergibt. marke1 bis markeN sind konstante Ausdrücke eines der oben angeführten Typen. Dabei kann marke1 durchaus ein String sein und marke2 eine Zahl. Variablen sind daher als Marken nicht zugelassen (sehr wohl natürlich als selector). Die Anweisungen, die einer Marke folgen, werden ausgeführt, wenn der Wert des selectors der Wert der Marke ist. Danach werden alle Anweisungen der Marken unterhalb der gefundenen Marke durchgeführt. Dies wird selten gewünscht. Die letzte An- weisungen hinter einer Marke ist daher `break`. `break` bewirkt, dass die Programmausführung hinter dem Switch fortgesetzt wird.

Ist der Wert des selectors in den Konstanten keiner Marke enthalten, wird der default-Zweig ausgeführt. Fehlt der default-Zweig, wird die switch-Anweisung ignoriert.

Die Zeile, die das Schlüsselwort switch enthält, wird nicht mit einem ; abgeschlossen.

Hinter einer Marke folgt ein Doppelpunkt (:).

Mit dem Switch können wir das Taschenrechner-Beispiel natürlich weitaus übersichtlicher implementieren. Dies zeigt Beispiel 6.13.

Beispiel 6.13 *Taschenrechner in php mit einem Switch*

```
<!--  Taschenrechner
  Dateiname: taschenrechner2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Taschenrechner </title>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='taschenrechner' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Erster Operand
                    </td>
                    <td>
                        <input type="text" name="ersterOperand" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Operator
                    </td>
                    <td>
                        <input type="text" name="operator" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Zweiter Operand
                    </td>
```

```
                    <td>
                        <input type="text" name="zweiterOperand" size=12>
                    </td>
                </tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        switch ($operator)
        {
            case "+":
                $ergebnis=$ersterOperand+$zweiterOperand;
                break;
            case "-":
                $ergebnis=$ersterOperand-$zweiterOperand;
                break;
            case "*":
                $ergebnis=$ersterOperand*$zweiterOperand;
                break;
            case "/":
                //Teilen durch 0 verhindern
                if($zweiterOperand!=0)
                {
                    $ergebnis=$ersterOperand/$zweiterOperand;
                }
                else
                {
                    $ergebnis="Versuch durch 0 zu teilen!";
                }
            break;
            default:
                $ergebnis="Falscher Operator eingegeben!";
        }
        echo "Das Ergebnis ist: $ergebnis";
    }//schliesst else zu if REQUEST_METHOD
?>
</body>
</html>
```

Jeder `switch` kann natürlich, wie ja auch Beispiel 6.12 zeigt, durch geschachtelte `if`-Anweisungen ersetzt werden. Andersrum ist dies nicht richtig. In den `case`-Marken einer `switch`-Anweisung kann nur auf Gleichheit getestet werden. Insbesondere logische Verknüpfungen wie in der Lösung zu Aufgabe 6.2 sind nicht möglich.

6.3.3 Beispiele

Volatilitäten (fortgesetzt)

Volatilität Bewertung kommt noch

Raketenbeispiel (fortgesetzt)

Wir erweitern unser Raketenbeispiel in Kapitel 6.2.4 folgendermaßen:

- Die Rakete startet und landet im gleichen Jahr und nicht mehr im gleichen Monat.
- Das Jahr ist kein Schaltjahr.
- Die Ausgabe soll allerdings bleiben, wie sie war. Wenn die Rakete mehrere Tage unterwegs ist, soll die Anzahl Tage mit ausgegeben werden, nicht jedoch die Anzahl Monate. Erfolgen Start und Landung jedoch am selben Tag, sollen Tage gar nicht auftauchen.

Auch hier müssen wir zur Problemlösung etwas überlegen. Zunächst gewärtigen wir uns noch einmal den Pseudocode des Algorithmus der Lösung zur Berechnung der Flugzeit(vgl. Kapitel 11):

1. Umrechnen der Startzeit in Sekunden
2. Umrechnen der Landezeit in Sekunden
3. Flugzeit in Sekunden = Landezeit in Sekunden - Startzeit in Sekunden
4. vorläufige Minuten der Flugzeit = Flugzeit in Sekunden integerdividiert durch 60
5. Sekunden der Flugzeit = Flugzeit in Sekunden modulo 60
6. Stunden der Flugzeit = vorläufige Minuten der Flugzeit integerdividiert durch 60
7. Minuten der Flugzeit = vorläufige Minuten der Flugzeit modulo 60

Eigentlich können wir diesen Algorithmus wiederverwenden. Es gibt nur ein kleineres Problem: Wie rechnen wir die Start-(oder Landezeit) in Sekunden um? Aber auch hiermit werden wir fertig.

Wir erinnert uns nämlich an unsere Problemlösung aus Kapitel 6.2.4. In der Lösung dort haben wir den Bezugspunkt verschoben. In der Lösung zu Aufgabe 5.3 hatten wir die Anzahl Sekunden betrachtet, die seit Beginn des Tages vergangen sind. In der Lösung in Kapitel 6.2.4 betrachten wir die Anzahl Sekunden, die seit Beginn des Monats vergangen sind. Auch das geht nun nicht mehr, weil die Rakete ja über Monatsgrenzen fliegen kann. Allerdings haben wir in unserer neuen Aufgabenstellung wieder eine Einschränkung. Die Rakete startet und landet im gleichen Jahr. Könnten wir ausrechnen, wieviele Tage seit Beginn des Jahres vergangen sind, wären wir durch mit der Problemlösung. Das ist aber gar nicht so schwer. Unsere Benutzer müssen ja den Monat des Abflugs und der Ankunft eingeben. Betrachten wir zur Vereinfachung nur den Startzeitpunkt. Wenn unsere Benutzer

- 1 eingeben (Monat Januar), haben wir die Lösung aus 6.2.4. Die Anzahl der Tage, die seit Beginn des Jahres vergangen sind, entsprechen dem Tagesdatum, das unsere Benutzer ja auch eingeben.
- 2 eingeben (Monat Februar), müssen wir zum Tagesdatum 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.

- 3 eingeben (Monat März), müssen wir zum Tagesdatum 31 + 28 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 4 eingeben (Monat April), müssen wir zum Tagesdatum 31 + 28 + 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 5 eingeben (Monat Mai), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 6 eingeben (Monat Juni), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 7 eingeben (Monat Juli), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 8 eingeben (Monat August), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 + 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 9 eingeben (Monat September), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 10 eingeben (Monat Oktober), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 +30 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 11 eingeben (Monat November), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 +30 + 31 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.
- 12 eingeben (Monat Dezember), müssen wir zum Tagesdatum 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 +30 + 31 + 30 hinzuzählen, um die Anzahl Tage seit Beginn des Jahres auszurechnen.

Durch diese Vorgehensweise haben wir den Bezugspunkt, an dem wir das Zählen der Sekunden starten, auf den Beginn des Jahres nach hinten verschoben.

Formulieren wir dies nun in Pseudocode. Wir gehen dabei davon aus, dass der eingegebene Tag des Starts auf starttag eingelesen wurde.
Einlesen des Startmonats
switch (startmonat)

- case 1: startTage=starttag
- case 2: startTage=31 + starttag
- case 3: startTage=31 + 28 + starttag
- case 4: startTage=31 + 28 + 31 + starttag
- case 5: startTage=31 + 28 + 31 + 30 + starttag
- case 6: startTage=31 + 28 + 31 + 30 + 31 + starttag
- case 7: startTage=31 + 28 + 31 + 30 + 31 + 30 + starttag
- case 8: startTage=31 + 28 + 31 + 30 + 31 + 30 + 31 starttag
- case 9: startTage=31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + starttag

- case 10: startTage=31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + starttag
- case 11: startTage=31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + starttag
- case 12: startTage=31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 + starttag

Nebenbei haben Sie hier gelernt, wie man einen Switch in Pseudocode abbildet. Nun sind wir aber soweit, unser neues Programm zu schreiben. Wir beginnen mit der Realisierung in JavaScript:

Beispiel 6.14 *Raketenprogramm Teil 3*

```
<-- Raketenbeispiel 3 des Textes
  Dateiname: raketen3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und sodann die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br>
    <script language = "JavaScript">
        var startSekunden;
        var startMinuten;
        var startStunden;
        var starttag;
        var startmonat;
        var startzeitInSekunden;
        var landeStunden;
        var landeMinuten;
        var landeSekunden;
        var landetag;
        var landemonat;
        var landezeitInSekunden;
        var flugzeitInSekunden;
        var flugzeitStunden;
        var flugzeitMinuten;
        var flugzeitSekunden;
        var flugzeitTage;
        //Einlesen
        startmonat=prompt("Bitte geben Sie den Startmonat ein","");
        starttag=prompt("Bitte geben Sie den Starttag ein","");
        startStunden=prompt("Bitte geben Sie die Stunden der Startzeit ein","");
        startMinuten=prompt("Bitte geben Sie die Minuten der Startzeit ein","");
        startSekunden=prompt("Bitte geben Sie die Sekunden der Startzeit ein","");
        landemonat=prompt("Bitte geben Sie den Landemonat ein","");
        landetag=prompt("Bitte geben Sie den Landetag ein","");
        landeStunden=prompt("Bitte geben Sie die Stunden der " +
                            "Landezeit ein","");
        landeMinuten=prompt("Bitte geben Sie die Minuten der " +
                            "Landezeit ein","");
        landeSekunden=prompt("Bitte geben Sie die Sekunden der " +
                            "Lande ein","");
```

```
//Umwandeln
startmonat=parseInt(startmonat);
starttag=parseInt(starttag);
startStunden=parseInt(startStunden);
startMinuten=parseInt(startMinuten);
startSekunden=parseInt(startSekunden);
landemonat=parseInt(landemonat);
landetag=parseInt(landetag);
landeStunden=parseInt(landeStunden);
landeMinuten=parseInt(landeMinuten);
landeSekunden=parseInt(landeSekunden);
// start-und landezeit in sekunden umrechnen
// zuerst der switch fuer startzeit
switch(startmonat)
{
    case 1:
        starttag=starttag; //ueberflussig, nur der Klarheit wegen
        break;
    case 2:
        starttag=31+starttag;
        break;
    case 3:
        starttag=31+28+starttag;
        break;
    case 4:
        starttag=31+28+31+starttag;
        break;
    case 5:
        starttag=31+28+31+30+starttag;
        break;
    case 6:
        starttag=31+28+31+30+31+starttag;
        break;
    case 7:
        starttag=31+28+31+30+31+30+starttag;
        break;
    case 8:
        starttag=31+28+31+30+31+30+31+starttag;
        break;
    case 9:
        starttag=31+28+31+30+31+30+31+31+starttag;
        break;
    case 10:
        starttag=31+28+31+30+31+30+31+31+30+starttag;
        break;
    case 11:
        starttag=31+28+31+30+31+30+31+31+30+31+starttag;
        break;
    case 12:
        starttag=31+28+31+30+31+30+31+31+30+31+30+starttag;
        break;
}
// nun landetage ausrechnen
```

```
switch(landemonat)
{
    case 1:
        landetag=landetag;//ueberflussig, nur der Klarheit wegen
        break;
    case 2:
        landetag=31+landetag;
        break;
    case 3:
        landetag=31+28+landetag;
        break;
    case 4:
        landetag=31+28+31+landetag;
        break;
    case 5:
        landetag=31+28+31+30+landetag;
        break;
    case 6:
        landetag=31+28+31+30+31+landetag;
        break;
    case 7:
        landetag=31+28+31+30+31+30+landetag;
        break;
    case 8:
        landetag=31+28+31+30+31+30+31+landetag;
        break;
    case 9:
        landetag=31+28+31+30+31+30+31+31+landetag;
        break;
    case 10:
        landetag=31+28+31+30+31+30+31+31+30+landetag;
        break;
    case 11:
        landetag=31+28+31+30+31+30+31+31+30+31+landetag;
        break;
    case 12:
        landetag=31+28+31+30+31+30+31+31+30+31+30+landetag;
        break;
}
// jetzt in Sekunden umrechnen
startzeitInSekunden=starttag*24*3600+
        startStunden*3600+startMinuten*60+startSekunden;
landezeitInSekunden=landetag*24*3600+
        landeStunden*3600+landeMinuten*60+landeSekunden;
// flugzeitInSekunden berechnen
flugzeitInSekunden=landezeitInSekunden-startzeitInSekunden;
if(flugzeitInSekunden<0)
{
    document.write("Fehleingabe: Landezeit vor Startzeit!");
}
else
{
    //Flugzeit umrechnen, zuerst Sekunden und Minuten
    flugzeitMinuten=Math.floor(flugzeitInSekunden/60);
```

```
            flugzeitSekunden=flugzeitInSekunden%60;
            //nun minuten und stunden
            flugzeitStunden=Math.floor(flugzeitMinuten/60);
            flugzeitMinuten=flugzeitMinuten%60;
            flugzeitTage=Math.floor(flugzeitStunden/24);
            flugzeitStunden=flugzeitStunden%24;
            //ausgeben
            if(flugzeitTage==0)
            {
                document.write("Die Flugzeit betr&auml;gt: <br>" +
                               flugzeitStunden + " Stunden <br>" +
                               flugzeitMinuten + " Minuten <br>" +
                               flugzeitSekunden + " Sekunden <br>");
            }
            else
            {
                document.write("Die Flugzeit betr&auml;gt: <br>" +
                               flugzeitTage + " Tage <br>" +
                               flugzeitStunden + " Stunden <br>" +
                               flugzeitMinuten + " Minuten <br>" +
                               flugzeitSekunden + " Sekunden <br>");
            }
        }
    </script>
</body>
</html>
```

Abb. 6.12 zeigt eine beispielhafte Anwendung des Programms.

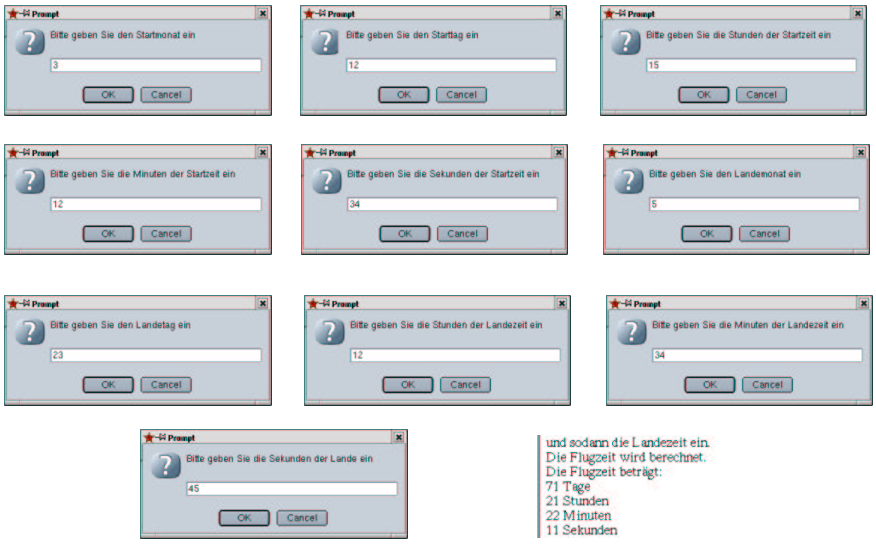


Abbildung 6.12: Das Raketenbeispiel Teil 3

Zum Abschluss die Realisierung in php und ein beispielhafter Durchlauf durch das php-Programm.

Beispiel 6.15 *Raketenbeispiel Teil 3 in php*

```
<!-- raketenbeispiel 3 des Textes
  Dateiname: raketen3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br> <hr>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren

        echo "<form name='raketen2' action='$PHP_SELF' method='post'>";
?>
        <table border>
            <tr>
                <td>
                    Startmonat
                </td>
                <td>
                    <input type="text" name="startmonat" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Starttag
                </td>
                <td>
                    <input type="text" name="starttag" size=12>
                </td>
            </tr>

            <tr>
                <td>
                    Startzeit Stunden
                </td>
                <td>
                    <input type="text" name="startStunden" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Startzeit Minuten
                </td>
                <td>
                    <input type="text" name="startMinuten" size=12>
                </td>
```

```
</tr>
<tr>
    <td>
        Startzeit Sekunden
    </td>
    <td>
        <input type="text" name="startSekunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landemonat
    </td>
    <td>
        <input type="text" name="landemonat" size=12>
    </td>
</tr>
<tr>
    <td>
        Landetag
    </td>
    <td>
        <input type="text" name="landetag" size=12>
    </td>
</tr>

<tr>
    <td>
        Landezeit Stunden
    </td>
    <td>
        <input type="text" name="landeStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landezeit Minuten
    </td>
    <td>
        <input type="text" name="landeMinuten" size=12>
    </td>
</tr>
<tr>
    <td>
        Landezeit Sekunden
    </td>
    <td>
        <input type="text" name="landeSekunden" size=12>
    </td>
</tr>
<tr>
    <td colspan="2" align="center">
        <input type="submit" name="Button1" value="Abschicken">
    </td>
```

```
            </tr>
        </table>
        </form>
<?php
    }
    else
    {
        // zweiter Aufruf nun rechnen
        // zunaechst starttage ausrechnen
        switch($startmonat)
        {
            case 1:
                $starttag=$starttag; //ueberflussig, nur der Klarheit wegen
                break;
            case 2:
                $starttag=31+$starttag;
                break;
            case 3:
                $starttag=31+28+$starttag;
                break;
            case 4:
                $starttag=31+28+31+$starttag;
                break;
            case 5:
                $starttag=31+28+31+30+$starttag;
                break;
            case 6:
                $starttag=31+28+31+30+31+$starttag;
                break;
            case 7:
                $starttag=31+28+31+30+31+30+$starttag;
                break;
            case 8:
                $starttag=31+28+31+30+31+30+31+$starttag;
                break;
            case 9:
                $starttag=31+28+31+30+31+30+31+31+$starttag;
                break;
            case 10:
                $starttag=31+28+31+30+31+30+31+31+30+$starttag;
                break;
            case 11:
                $starttag=31+28+31+30+31+30+31+31+30+31+$starttag;
                break;
            case 12:
                $starttag=31+28+31+30+31+30+31+31+30+31+30+$starttag;
                break;
        }
        // nun landetage ausrechnen
        switch($landemonat)
        {
            case 1:
                $landetag=$landetag;//ueberflussig, nur der Klarheit wegen
                break;
```

```
        case 2:
            $landetag=31+$landetag;
            break;
        case 3:
            $landetag=31+28+$landetag;
            break;
        case 4:
            $landetag=31+28+31+$landetag;
            break;
        case 5:
            $landetag=31+28+31+30+$landetag;
            break;
        case 6:
            $landetag=31+28+31+30+31+$landetag;
            break;
        case 7:
            $landetag=31+28+31+30+31+30+$landetag;
            break;
        case 8:
            $landetag=31+28+31+30+31+30+31+$landetag;
            break;
        case 9:
            $landetag=31+28+31+30+31+30+31+31+$landetag;
            break;
        case 10:
            $landetag=31+28+31+30+31+30+31+31+30+$landetag;
            break;
        case 11:
            $landetag=31+28+31+30+31+30+31+31+30+31+$landetag;
            break;
        case 12:
            $landetag=31+28+31+30+31+30+31+31+30+31+30+$landetag;
            break;
    }

    $startzeitInSekunden=$starttag*24*3600+
                $startStunden*3600+$startMinuten*60+$startSekunden;
    $landezeitInSekunden=$landetag*24*3600+
                $landeStunden*3600+$landeMinuten*60+$landeSekunden;
    // flugzeitInSekunden berechnen
    $flugzeitInSekunden=$landezeitInSekunden-$startzeitInSekunden;
    if($flugzeitInSekunden<0)
    {
        echo "Fehleingabe: Landezeit vor Startzeit!";
    }
    else
    {
        //Flugzeit umrechnen, zuerst Sekunden und Minuten
        $flugzeitMinuten=floor($flugzeitInSekunden/60);
        $flugzeitSekunden=$flugzeitInSekunden%60;
        //nun minuten und stunden
        $flugzeitStunden=floor($flugzeitMinuten/60);
        $flugzeitMinuten=$flugzeitMinuten%60;
        $flugzeitTage=floor($flugzeitStunden/24);
```

```
            $flugzeitStunden=$flugzeitStunden%24;
            //ausgeben
            if($flugzeitTage==0)
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
            else
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitTage Tage <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
        }
    }
?>
</body>
</html>
```

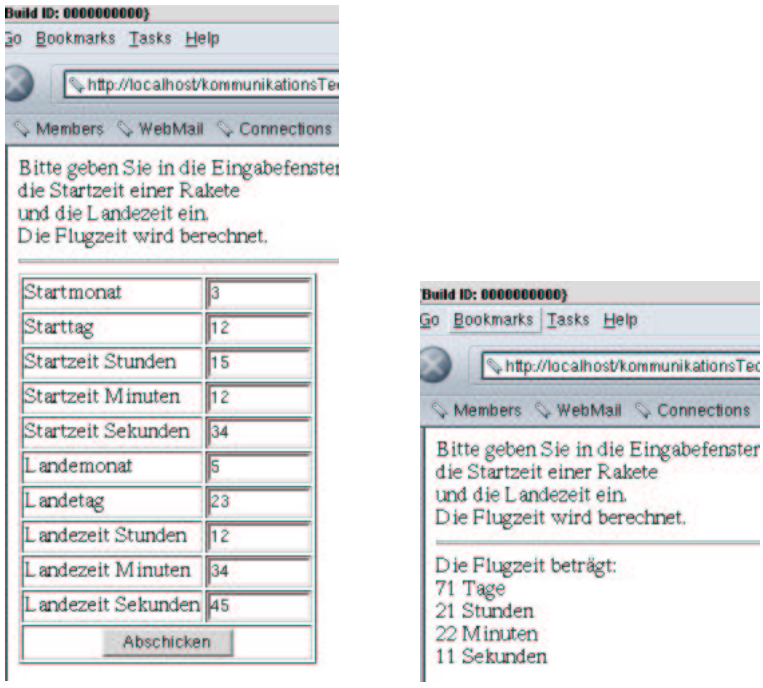


Abbildung 6.13: Das Raketenbeispiel Beispiel Teil 3 in php

Aufgabe 6.3 *Implementieren Sie das Taschenrechnerprogramm aus Beispiel* **??** *in JavaScript.*

Aufgabe 6.4 *Implementieren Sie das Taschenrechnerprogramm aus Beispiel 6.13 in JavaScript.*

Kapitel 7

Funktionen

7.1 Motivation

Betrachten wir noch einmal Beispiel 6.15[1]. In diesem Beispiel befinden sich zwei große `Switch`-Konstrukte. Sie sind im Wesentlichen identisch, der eine `Switch` berechnet die Tage im Jahr, die seit dem Start vergangen sind, der andere tut dasselbe für den Landezeitpunkt. Dies vergrößert unser Programm unnötig und macht es auch fehleranfälliger. Denn gerade bei einer so großen Konstruktion, wie bei diesen beiden Switchen, kann man sich leicht mal vertippen. Hat man den zweiten Switch vermittels kopieren und einfügen und dann starttag gegen landetag ersetzen, erzeugt, ist der Fehler zweimal im Programm. Hat man hingegen jeden Switch einzeln getippt[2], funktioniert der Algorithmus in dem einen Switch, im anderen nicht, oder, schlimmer noch, man hat sich vielleicht zweimal vertippt und dann in jedem `Switch` einen anderen Fehler.

Besser wäre, es gäbe eine Möglichkeit, in unserer Programmiersprache[3] die Vorgehensweise zur Berechnung der vergangenen Tage zu implementieren. Diese könnten wir dann einmal für den Starttag und einmal für den Landetag aufrufen.

Betrachten wir ein weiteres Beispiel: Die Seiten des Intranet der FH-Bochum werden, wie Sie ja wissen, dynamisch mit php und Javascript erzeugt. Jede Seite enthält eine kurze Überschrift, die die jeweils ausgewählte Funktionalität beschreibt (vgl. Abb. 7.1).

Diese Überschrift wird durch folgenden html-Code erzeugt[4]:

```
<table border="0" align="center" width=620>
    <tr>
    <td>
            <div class="liste3">
                Auswahl der Vorlesungspl&auml;ne
            </div>
        </td>
    </tr>
</table>
```

Hier variiert nur der Text, der ausgegeben wird. In Abb. 7.1 war es “Auswahl der Vorlesungspläne”. Wählen Sie hingegegen die Ausfall-Funktion, ist die Überschrift: “Vorlesungsausfälle im Fachbereich” und der erzeugende html-Code ist:

[1] Wir hätten natürlich auch Beispiel 6.14 betrachten können :-).

[2] können eigentlich nur Leute, die das 10-Finger System beherrschen, ernsthaft in Erwägung ziehen.

[3] so wie im Pseudocode

[4] Selbstverständlich muss es eine Stylesheet-Datei, die das Format “liste3” definiert, geben.

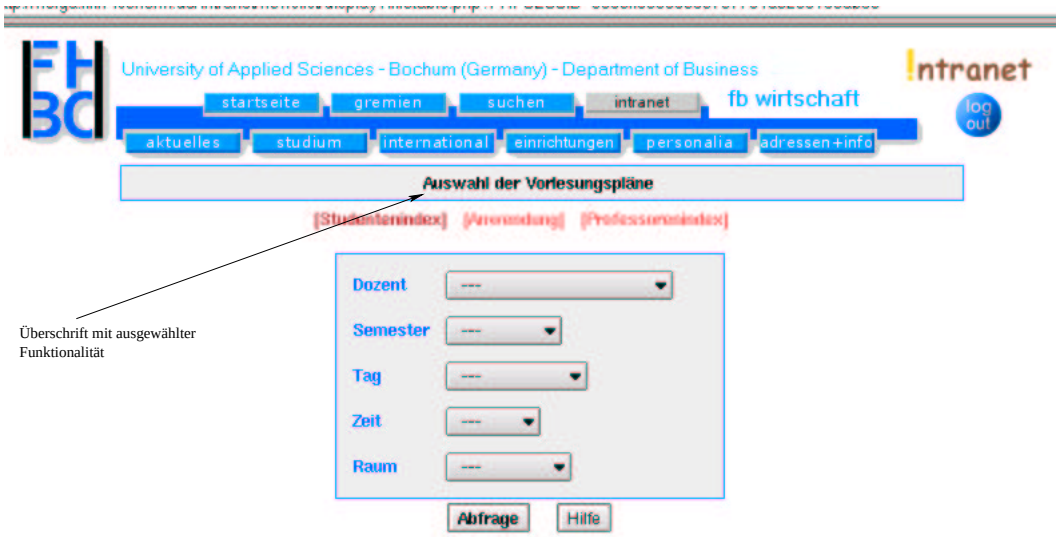


Abbildung 7.1: Überschrift mit Funktionalität

```
<table border="0" align="center" width=620>
    <tr>
        <td>
            <div class="liste3">
                Vorlesungsausf&auml;lle im Fachbereich
            </div>
        </td>
    </tr>
</table>
```

Solcher Code würde nun in allen unseren Seiten stehen. Wollten wir etwas ändern, z.B. eine Tabelle mit zwei Spalten erzeugen, müßte jede Seite des Intranet angefasst und angepasst werden. Besser wäre, wir könnten etwas programmieren, wo wir sagen könnten: Dies ist der Text, der als Funktionalitätsüberschrift ausgegeben werden soll, gib den jetzt so aus.

JavaScript und php bieten uns, wie (fast) alle Programmiersprachen, eine Möglichkeit, oben beschriebene Problematiken zu lösen. Wir können Anweisungsfolgen zu Funktionen zusammenfassen und diesen Funktionen einen Namen geben. Für Funktionsnamen gelten die im Kapitel für Variablennamen dargestellten Regeln. Funktionen beginnen mit dem Schlüsselwort `function`, dann folgt der Name der Prozedur, dann eventuelle Übergabeparameter[5].

Bevor wir das Funktionen-Konzept theoretisch behandeln, schauen wir uns ein Beispiel an:

Beispiel 7.1 *Funktionen, die Erste*

```
<!-- Das Programm zur Funktionen-Demonstration
  Dateiname: funktionen1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>
```

[5]Übergabeparameter werden gleich erklärt.

```
        Funktionen zum Ersten
  </title>
  <LINK rel="STYLESHEET" type="TEXT/CSS"
                href="/intranet/styleSheets/standard.css">
<?php
    function darstellenAlsListe3($text)
    {

        echo "<table border='0' align='center' width=620>";
        echo "        <tr>";
        echo "                <td>";
        echo "                        <div class='liste3'>";
        echo "                            $text";
        echo "                        </div>";
        echo "                </td>";
        echo "        </tr>";
        echo "</table>";
    }
?>

</head>
<body>
    <h2>
        Demonstration des Funktionskonzepts:
<?php
    darstellenAlsListe3("Der erste Text!");
    darstellenAlsListe3("Der zweite Text!");
    darstellenAlsListe3("Der dritte Text!");
?>
</body>
</html>
```

Dies führt zu der in Abb. 7.2 gezeigten Darstellung:

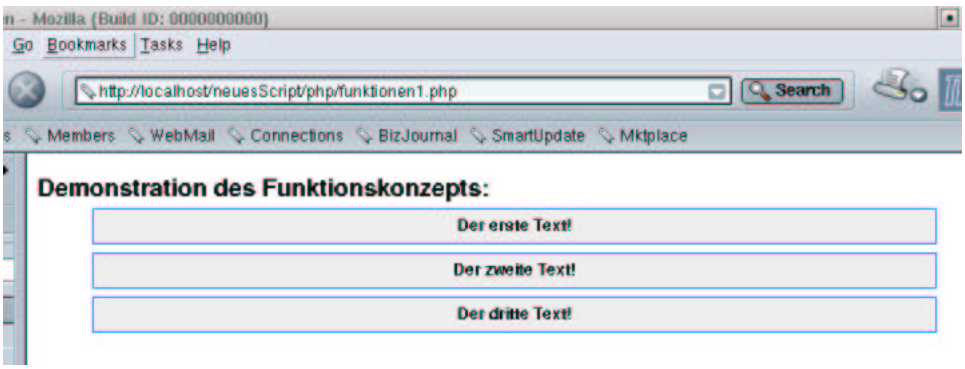


Abbildung 7.2: Funktionen die Erste auf dem Bildschirm

Funktionen werden zuerst deklariert und implementiert. Dies erfolgt in Beispiel 7.1 im `head`-Teil der html-Datei. Eine Funktions-Deklaration besteht aus dem reservierten Wort `function`, dem

Namen der Funktion (im Beispiel darstellenAlsListe3), runden Klammern und den Übergabeparametern[6] (in unserem Fall nur einer: $text). Dies wird häufig auch als Signatur der Funktion bezeichnet.

Danach folgt die Implementierung der Funktion. Dies sind Kommandos der Programmiersprache, eingeschlossen in geschweiften Klammern.

Funktionen werden dann aufgerufen (oder einfach gerufen). Dies erfolgt in Beispiel 7.1 im `body`-Teil der html-Datei. Der Funktionsaufruf erfolgt über den Namen der Funktion, gefolgt von runden Klammern und den aktuellen Werten der Übergabeparameter.

Beim Aufruf der Funktion erhalten die Übergabeparameter in der Deklaration der Funktion die aktuellen Werte aus dem Aufruf zugewiesen. Durch den ersten Aufruf

```
darstellenAlsListe3("Der erste Text!");
```

werden die im `head`-Teil definierten Anweisungen der Funktion durchgeführt. Vorher wird der Übergabevariable $text (Übergabevariable ist ein anderes Wort für Übergabeparameter) der Wert ”Der erste Text!” zugewiesen. An Abb. 7.2 sehen wir, dass sich dies auch so verhält: ”Der erste Text!” erscheint in einer Tabellenzelle im Format “liste3”.

Die beiden anderen Aufrufe der Funktion funktionieren völlig analog.

Wollen wir die Ausgabe ändern, beispielsweise dadurch, dass die Tabelle einen sichtbaren Rand bekommen soll und in einer eigenen Zelle “Intelligenteste Texte” vor dem jeweils auszugebenden Text erscheint, müssen wir nur die Funktion verändern:

Beispiel 7.2 *Funktionen, die Zweite (Geändertes Layout)*

```
<!-- Das Programm zur Funktionen-Demonstration
  Dateiname: funktionen2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>
        Funktionen zum Zweiten
  </title>
  <LINK rel="STYLESHEET" type="TEXT/CSS"
                href="/intranet/styleSheets/standard.css">
<?php
    function darstellenAlsListe3($text)
    {
        echo "<table border='3' align='center' width=620>";
        echo "        <tr>";
        echo "                <td>";
        echo "Intelligenteste Texte:";
        echo "                </td>";
        echo "                <td>";
        echo "                        <div class='liste3'>";
        echo "$text";
        echo "                        </div>";
        echo "                </td>";
        echo "        </tr>";
        echo "</table>";
    }
?>
```

[6] Übergabeparameter sind nicht zwingend notwendig, wir werden Beispiele ohne Übergabeparameter sehen, die runden Klammern sind immer nötig.

```
</head>
<body>
    <h2>
        Demonstration des Funktionskonzepts:
<?php
    darstellenAlsListe3("Der erste Text!");
    darstellenAlsListe3("Der zweite Text!");
    darstellenAlsListe3("Der dritte Text!");
?>
</body>
</html>
```

Dies führt zu der in Abb. 7.3 gezeigten Darstellung:

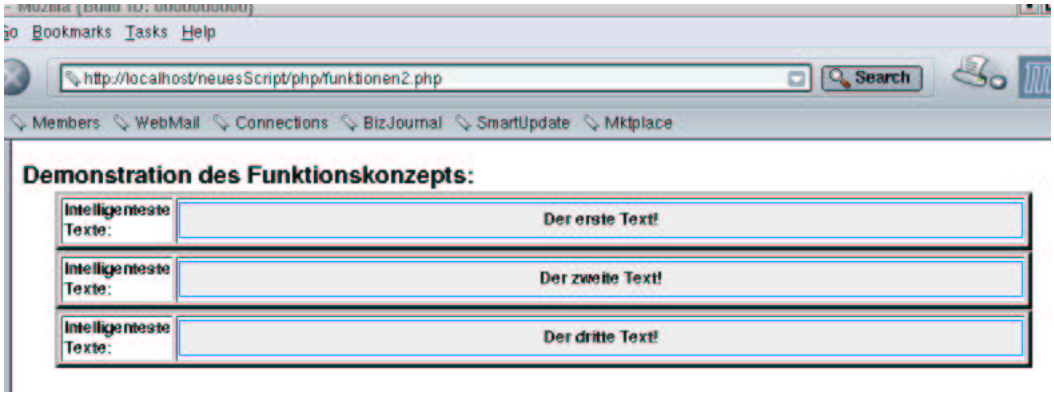


Abbildung 7.3: Funktionen die Zweite auf dem Bildschirm

Bisher haben wir durch den Einsatz von Funktionen Schreibarbeit gespart (wir haben die Funktion deklariert und dann dreimal genutzt und daher die Anweisugen, die zur Bildschirmausgabe führen nur einmal in der html-Datei einfügen müssen) und es gibt eine eine zentrale Stelle in der Datei, in der wir die Ausgabe bestimmen (ohne Funktion hätten wir die Änderung der Tabelle und des Textes dreimal durchführen müssen[7]).

Der eigentliche Clou besteht aber nun in der Auslagerung der Funktion in eine eigene Datei. Wir erstellen eine Datei für die Darstellungsfunktionen und definieren unsere Funktion dort. Dann binden wir die Datei, wie in Kapitel 3 gelernt, in unsere html-Datei ein. Wir können die Funktion nun benutzen, als ob sie in der Datei selber definiert wäre:

Beispiel 7.3 *Funktionen, die Dritte: Zuerst die Datei mit der Funktionsdefinition*

```
<?php
    // Sammlung der Darstellungsfunktionen
    // Dateiname: darstellungsFunktionen.inc.php
    // Verzeichnis: includes
    function darstellenAlsListe3($text)
    {
```

[7] Wobei wir die Chance, einen Fehler zu machen, natürlich ebenfalls verdreifachen.

```
        echo "<table border='3' align='center' width=620>";
        echo "        <tr>";
        echo "                <td>";
        echo "                      Intelligenteste Texte:";
        echo "                </td>";
        echo "                <td>";
        echo "                        <div class='liste3'>";
        echo "                            $text";
        echo "                        </div>";
        echo "                </td>";
        echo "        </tr>";
        echo "</table>";
    }
?>
```

Beispiel 7.4 *Funktionen, die Dritte: Der Aufruf*

```
<!-- Das Programm zur Funktionen-Demonstration
  Dateiname: funktionen3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>
        Funktionen zum Dritten
  </title>
  <LINK rel="STYLESHEET" type="TEXT/CSS"
                    href="/intranet/styleSheets/standard.css">
<?php
    require_once("./includes/darstellungsFunktionen.inc.php");
?>
</head>
<body>
    <h2>
        Demonstration des Funktionskonzepts:
<?php
    darstellenAlsListe3("Der erste Text!");
    darstellenAlsListe3("Der zweite Text!");
    darstellenAlsListe3("Der dritte Text!");
?>
</body>
</html>
```

An der Ausgabe ändert sich selbstverständlich nichts.

Diese Vorgehensweise hat diverse Vorteile:

- Eine einmal geschriebene Funktion kann in beliebig viele Dateien eingebunden werden. Änderungen der Funktion sind in den die Funktion einbindenden Dateien sofort sichtbar.
- Funktionen können ohne weiteres von anderen geschrieben werden. Wir müssen nur den Dateinamen kennen (um die Datei einbinden zu können) und die Signatur der Funktion (um sie aufrufen zu können).

7.2 Syntax

Funktionen werden durch das Schlüsselwort `function` definiert. Danach folgt der Name der Funktion. Die Übergabeparameter einer Funktion folgen in runden Klammern. Mehrere Übergabeparameter werden durch Kommata getrennt. Funktionen ohne Übergabeparameter erhalten in der Deklaration leere runde Klammern.

Der Rumpf der Funktion schließt sich (eingeschlossen in geschweifte Klammern) an.

```
function nameDerFunktion(Parameter1, Parameter2, Parameter)
{
    anweisung1;
    anweisung2;
       ...
    anweisungN;
}
```

Die Übergabeparameter sind Variablen (müssen sie sein, da sie bei jedem Aufruf der Funktion andere Werte annehmen).

Funktionen werden aus unseren Programmen heraus aufgerufen.

Das Arbeiten mit Funktionen erfolgt also in zwei Schritten. Zunächst wird die Funktion deklariert und implementiert. Unter Deklarieren verstehen wir, der Funktion einen Namen zu geben und das Schlüsselwort `function` gefolgt von diesem Namen und in runden Klammern den Übergabeparametern, aufzuführen.

Unter Implementieren verstehen wir das Aufführen der Anweisungen der Funktion zwischen den geschweiften Klammern.

Deklaration und Implementierung einer Funktion erfolgen genau einmal (pro Funktion).

Der zweite Schritt ist der Aufruf der Funktion. Der Aufruf erfolgt, wie wir schon gesehen haben, über den Namen der Funktion. Eine Funktion kann beliebig oft aus beliebig vielen Programmen aufgerufen werden.

Funktionen können Rückgabewerte haben. Der Wert, den eine Funktion zurückgibt, folgt nach dem Schlüsselwort `return`. Nach `return` kehrt die Funktion zurück, Code-Zeilen nach `return` werden nicht mehr ausgeführt.

Auch dies machen wir uns an einem kleinen Beispiel klar: Wir wollen eine Anwendung schreiben, die es Vertriebspartnern ermöglicht, Provisionen über Umsätze über das Internet einzusehen. Dazu geben die Vertriebspartner ihren bisherigen Umsatz ein, wir geben die Provision in einer html-Seite aus. Ist der Umsatz größer als 10000 Euro, gibt es 10% Provision, sonst gibt es keine Provision[8]. Ist der eingegebene Umsatz kleiner als Null, soll „Fehler“ zurückgegeben werden. Die Provisionsberechnung erfolgt in einer Funktion (diesmal machen wir das in JavaScript).

Beispiel 7.5 *Funktion mit Rückgabewert*

```
<!-- Programm zur Nutzung von Rueckgabewerten
  Dateiname: funktionen1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Provisionen 1</title>
  <script language="JavaScript">
        function berechneProvision(umsatz)
```

[8] Natürlich kann man das auch im Kopf berechnen, aber der gute Wille zählt.

```
        {
            var provision;
            if(umsatz<=0)
            {
                return "Fehler";
            }
            if(umsatz<10000)
            {
                provision=0;
            }
            else
            {
                provision=umsatz*0.1;
            }
            return provision;
        }
  </script>
</head>
<body>
    <h2> Demonstration von return </h2>
    <script language="JavaScript">
        var umsatz;
        var provision;
        umsatz=prompt("Bitte geben Sie den Umsatz ein!");
        umsatz=parseFloat(umsatz);
        provision=berechneProvision(umsatz);
        document.write("Die Provision zu " + umsatz +
                       " beträgt " + provision + " !")
    </script>
</body>
</html>
```

Abb. 7.4 zeigt beispielhafte Ein- und Ausgaben von Beispiel 7.5[9].

Die Deklaration der Funktion erfolgt wieder im `head`-Teil der html-Datei[10]. Die Zeile

```
function berechneProvision(umsatz)
```

entspricht dem bisher bekannten: Die Funktion heißt berechneProvision. Sie erwartet einen Übergabeparameter. Dieser heißt innerhalb der Funktion umsatz. Danach wird eine Variable (provision) deklariert. Der nächste Teil innerhalb des `if`-Blocks ist neu:

```
if(umsatz<=0)
{
    return "Fehler";
}
```

Wenn die Funktion also mit einem Übergabeparameter aufgerufen wurde, dessen Wert kleiner gleich Null ist, ergibt die Bedingung der `if`-Anweisung true. In diesem Fall wird in den `if`-Teil verzweigt. Der `if`-Teil beinhaltet eine einzige Anweisung. Durch diese Anweisung gibt die Funktion den String

[9] Die Fehlerausgabe ist ja nicht besonders hübsch, aber wir wollen das Beispiel ja einfach halten.

[10] Selbstverständlich sollten wir, wenn wir es ernst meinen mit Funktionen, diese in eigene Dateien auslagern!

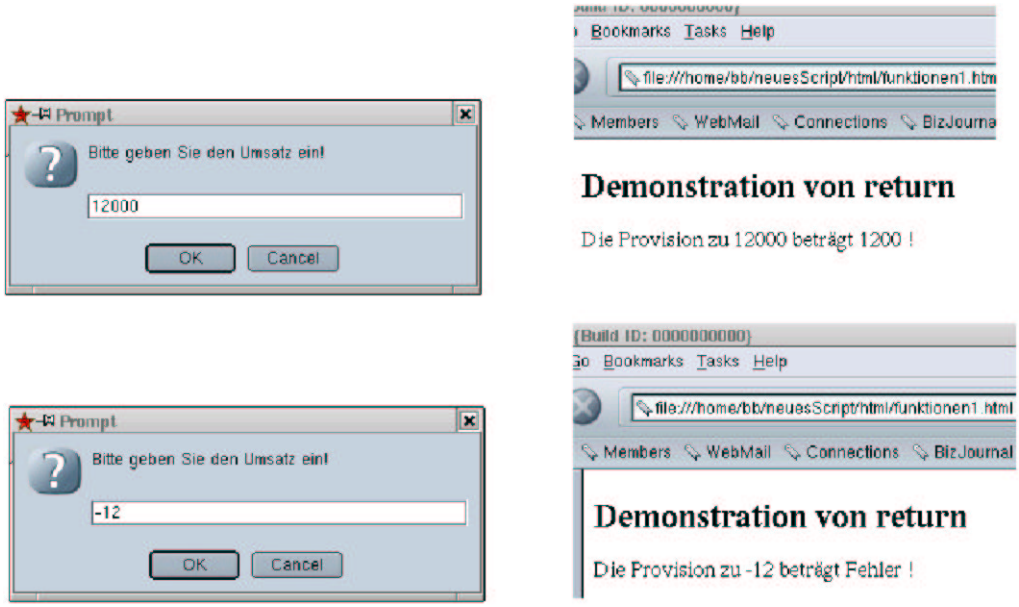


Abbildung 7.4: Überschrift mit Funktionalität

"Fehler" zurück und beendet sich. Keine der darunterliegenden Anweisungen wird ausgeführt. In Informatik-Deutsch sagt man, die Kontrolle geht an das aufrufende Programm zurück.

Ist der Wert des Übergabeparameters größer als Null, wird die `if`-Anweisung ignoriert (kein `else`-Teil vorhanden. Das Programm berechnet in diesem Fall die Provision:

```
if(umsatz<10000)
{
    provision=0;
}
else
{
    provision=umsatz*0.1;
}
```

Zum Abschluss gibt das Programm die berechnete Provision zurück und beendet sich[11]:

```
return provision;
```

Nun stellt sich die Frage: Wie erhält das Hauptprogramm den Rückgabewert? Auch darauf gibt Beispiel 7.5 die Antwort. Funktionen, die einen Wert zurückgeben, stehen auf der rechten Seite einer Zuweisung und schreiben den Rückgabewert somit auf eine Variable des Hauptprogramms[12]:

```
provision=berechneProvision(umsatz);
```

[11] Hier wäre die Funktion sowieso zu Ende gewesen, weil die `return`-Anweisung die letzte Anweisung der Funktion ist. Wie wir ja bereits gesehen haben, hören Funktionen mit der schließenden Klammer sowieso auf.

[12] Funktionen mit Rückgabewerten können ohne weiteres auch alleine auf einer Zeile des Programms stehen, wie die Funktionen ohne Rückgabewerte. Der Rückgabewert verschwindet dann einfach im Datennirwana. Doch das ist selten so gewollt.

Nun erkennen wir auch die ganze Wahrheit: Wir haben die ganze Zeit schon mit Funktionen[13] gearbeitet. `floor` in php, `Math.floor` oder `parseFloat` in JavaScript[14] sind Funktionen. Die haben nur nicht wir programmiert, sondern die Entwickler von php bzw. JavaScript und wir können sie benutzen. In Wahrheit gibt es in php und auch in JavaScript hunderte von Funktionen, die andere geschrieben haben und die wir einfach benutzen können. Schauen Sie einfach mal in die Funktionsreferenz von php. Abb. 7.5 zeigt beispielhaft die Funktionen, die php zur Anbindung von MySql-Datenbanken zur Verfügung stellt.

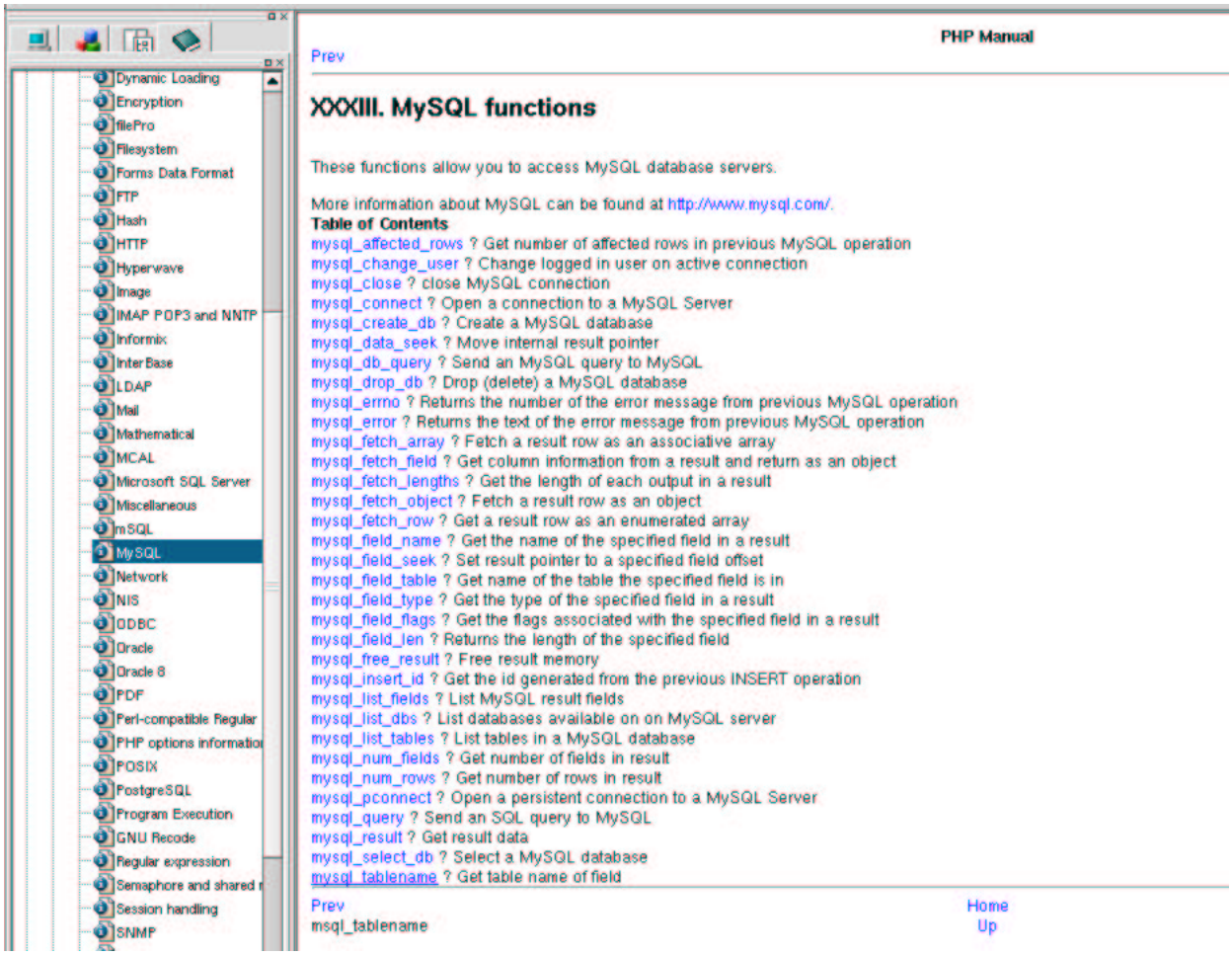


Abbildung 7.5: Funktionsreferenz MySql-Funktionen in php

Die Übergabeparameter im Kopf der Funktionsdeklaration heißen formale Parameter, die Variablen oder Konstanten beim Aufruf heißen aktuelle Parameter.

Formale Parameter und aktuelle Parameter müssen nicht den gleichen Namen erhalten. Die Übergabe erfolgt anhand der Reihenfolge. Der erste Aktualparameter wird an den ersten Formalparameter, der zweite Aktualparameter an den zweiten Formalparameter, usw. übergeben[15]. Übereinstimmen muss allerdings die Anzahl Parameter[16]. Beispiel 7.6 ist also eine identische Alternative:

[13] Hatte ich ja auch schon immer mal angemerkt!
[14] Dies sind zwar eigentlich Methoden, doch der Unterschied ist eher akademisch. Ich erkläre ihn in Kapitel 8.
[15] War bisher noch kein Problem, unsere Funktionen hatten jeweils nur einen Übergabeparameter
[16] Auch das ist nicht ganz richtig, wie wir später sehen werden.

Beispiel 7.6 *Funktion mit Rückgabewert (verschiedene Variablennamen)*

```
<!-- Programm zur Nutzung von Rueckgabewerten
  Dateiname: funktionen2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Provisionen 1</title>
  <script language="JavaScript">
        function berechneProvision(umsatzFunktion)
        {
            var provisionFunktion;
            if(umsatzFunktion<=0)
            {
                return "Fehler";
            }
            if(umsatzFunktion<10000)
            {
                provisionFunktion=0;
            }
            else
            {
                provisionFunktion=umsatzFunktion*0.1;
            }
            return provisionFunktion;
        }
  </script>
</head>
<body>
    <h2> Demonstration von return </h2>
    <script language="JavaScript">
        var umsatz;
        var provision;
        umsatz=prompt("Bitte geben Sie den Umsatz ein!","");
        umsatz=parseFloat(umsatz);
        provision=berechneProvision(umsatz);
        document.write("Die Provision zu " + umsatz +
                       " beträgt " + provision + " !")
    </script>
</body>
</html>
```

Die Ausgabe entspricht natürlich Abb. 7.4, da Beispiele 7.6 und 7.5 identisch sind.

7.3 Beispiele

7.3.1 Euro-Dollar-Umrechnung (fortgesetzt)

Unser Programm zur Euro-Dollar-Umrechnung soll weiter verbessert werden. Das Programm soll flexibler einsetzbar werden. Wir wollen weiterhin als eigenständige Dienstleistung anbieten, Dollar in Euro und umgekehrt, umzurechnen. Andererseits sind die Preise auf unseren Produktseiten nur in

Euro angegeben. Wir wollen aber nun auch im amerikanischen Markt tätig werden, daher sollen die Preise auch dort in Dollar erscheinen. Zudem wollen wir uns die Alternative offenhalten, die Basis der Preise auf Dollar umzustellen und dann in Euro umzurechnen.

Die Lösung liegt nach dem bisher gelernten auf der Hand: Wir werden die Euro-Dollar-Umrechnung in eine Funktion auslagern. Die Funktion wird in eine eigene Datei ausgelagert und in die html-Seite zur Euro-Dollar-Umrechnung eingebunden. Andere Seiten können unsere Funktion dann ebenfalls einbinden und nutzen. Wir starten mit der php-Lösung.

Beispiel 7.7 *Euro zum Dritten: Die ausgelagerte Berechnungsdatei*

```
<?php
// Funktion zur Dollar-Euro oder Euro-Dollar Umrechnung
// Datei:euroDollarUmrechnung.inc.php
// Verzeichnis: includes
    function euroDollarUmrechnung($zielwaehrung, $betrag)
    {
        $kurs=0.9;
        if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
        {
            $dollarbetrag=$kurs*$betrag;
            return "$dollarbetrag Dollar";
        }
        if(($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
        {
            $eurobetrag=(1/$kurs)*$betrag;
            return  "$eurobetrag Euro";
        }
    }
?>
```

Beispiel 7.8 *Euro zum Dritten: Der Aufruf*

```
<!-- Programm zur Euro-Dollar Umrechnung Teil3
  Dateiname: euro3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Teil 3</title>
<?php
    require_once("./includes/euroDollarUmrechnung.inc.php");
?>
</head>
<body>
<?php
    // Wir pruefen zuerst, ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='euro2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
```

```
                    <td>
                        Zielw&auml;hrung
                    </td>
                    <td>
                        <input type="text" name="zielwaehrung" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Betrag
                    </td>
                    <td>
                        <input type="text" name="betrag" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
            if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar") ||
               ($zielwaehrung=="euro")||($zielwaehrung=="Euro"))
            {
                $ergebnis=euroDollarUmrechnung($zielwaehrung,$betrag);
                echo "Ihre Eingabe entspricht $ergebnis!";
            }
            else
            {
                echo("Falsche Zielw&auml;hrung: <br>" .
                        "Erlaubt sind: Euro oder Dollar!");
            }
    }
?>
</body>
</html>
```

Die Funktion zur Euro-Dollar-Umrechnung erwartet zwei Parameter: Die Zielwährung und den umzurechnenden Betrag:

```
function euroDollarUmrechnung($zielwaehrung, $betrag)
```

Beachten Sie, dass damit auch die Reihenfolge bei jedem Aufruf der Funktion festgelegt ist: Als erster Aktualparameter muss die Zielwährung erscheinen, als zweiter Parameter der umzurechnende Betrag.

Die Implementierung besteht nach der Festlegung des Kurses aus zwei `if`-Anweisungen: Hier wird jeweils überprüft, wohin umgerechnet wird, dann wird die Umrechnung durchgeführt und der berechnete Wert zurückgegeben.

Beim Aufruf (Beispiel 7.8) wird zunächst im `head`-Teil der html-Datei die die Funktion enthaltende Datei mit `require_once` eingebunden. Beachten Sie, diese Datei liegt im Verzeichnis includes, welches sich unterhalb des Verzeichnisses befindet, in dem die aufrufende Datei beheimatet ist. Der erste Teil (`REQUEST_METHOD!=POST`) entspricht Beispiel 6.8. Das Eingabeformular wird aufgebaut. Beim zweiten Aufruf der Datei wird nun nur noch geprüft, ob die Zielwährung okay ist, dann wird die Funktion aufgerufen und das Ergebnis ausgegeben:

```
$ergebnis=euroDollarUmrechnung($zielwaehrung,$betrag);
echo "Ihre Eingabe entspricht $ergebnis!";
```

Auch hier heißen die Variablen im Hauptprogramm und in der Funktion gleich. Das liegt aber nur daran, dass “zielwaehrung” und “betrag” eigentlich ganz gute Namen für Dinge, wie Zielwährung und Betrag sind. Für die Programme spielt das keine Rolle. Es zählt die Reihenfolge:

```
Deklaration:
   function euroDollarUmrechnung($zielwaehrung, $betrag)
Aufruf:
   $ergebnis=euroDollarUmrechnung($zielwaehrung,$betrag);
```

Erfolgt der Aufruf in der falschen Reihenfolge, wie in Beispiel 7.9, ist ein falsches Ergebnis die Folge. Dies zeigt Abb.7.6.

Beispiel 7.9 *Euro zum Vierten: Falscher Aufruf*

```
<!-- Programm zur Euro-Dollar Umrechnung Teil 4
    Falsche Reihenfolge der Übergabeparameter
  Dateiname: euro4.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Teil 3</title>
<?php
    require_once("./includes/euroDollarUmrechnung.inc.php");
?>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='euro2' action='$PHP_SELF' method='post'>";
?>
           <table border>
               <tr>
                   <td>
                       Zielw&auml;hrung
                   </td>
                   <td>
                       <input type="text" name="zielwaehrung" size=12>
                   </td>
               </tr>
               <tr>
                   <td>
```

```
                        Betrag
                    </td>
                    <td>
                        <input type="text" name="betrag" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
            if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar") ||
               ($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
            {
                // jetzt kommt der Fehler, Reihenfolge verdreht
                $ergebnis=euroDollarUmrechnung($betrag, $zielwaehrung);
                echo "Ihre Eingabe entspricht $ergebnis!";
            }
            else
            {
                echo("Falsche Zielw&auml;hrung: <br>" .
                        "Erlaubt sind: Euro oder Dollar!");
            }
    }
?>
</body>
</html>
```

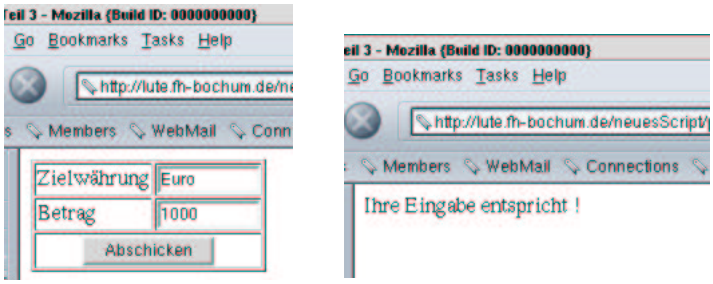


Abbildung 7.6: Ausgabe von Beispiel 7.9

Die Ausgabe in Abb.7.6 ist aber leicht erklärt. In Beispiel 7.9 ist beim Aufruf die Reihenfolge der Übergabeparameter vertauscht.

```
$ergebnis=euroDollarUmrechnung($betrag, $zielwaehrung);
```

Daher erhält die Variable $zielwaehrung der Funktion den Wert der Variablen $betrag des Hauptprogramms zugewiesen. Dies ist aber, wie man Abb.7.6 entnehmen kann, 1000. Dies bedeutet, der Wert von $zielwaehrung in der Funktion “euroDollarUmrechnung” ist nach dem Aufruf 1000.

In den beiden `if`-Anweisungen wird der Wert von $zielwaehrung mit “Dollar”, “dollar”, “Euro” und “euro” verglichen. Da der Wert von $zielwaehrung bei diesem Aufruf aber 1000 ist, trifft der Vergleich in keinem Fall zu. Die Funktion macht also nichts und gibt daher auch nichts zurück. Und darum wird vom Hauptprogramm auch nur die Zeichenkette “Ihre Eingabe entspricht” ausgegeben, da die Variable $ergebnis ja mit nichts besetzt ist.

Nun sieht dieses Beispiel ziemlich akademisch aus, weil Eingabeüberprüfungen ja im Hauptprogramm stattfinden und die Funktion eigentlich nur umrechnet. Unser Programm ist im Gegensatz zu Beispiel 6.8 nicht wesentlich kürzer.

Doch betrachten wir folgende weitere Anwendung von Beispiel 7.7. Wir wollen, wie in der Aufgabenstellung beschrieben, unsere Funktion auf den Produktseiten unseres Unternehmens einsetzen:

Beispiel 7.10 *Produktseite: Nutzung der Umrechnungsfunktion*

```
<!-- Programm zur Anwendung der Euro-Dollar Umrechnung
  Dateiname: produkte1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Sanit&auml;rartikel</title>
<?php
    require_once("./includes/euroDollarUmrechnung.inc.php");
?>
</head>
<body>
<table border="1">
  <tr>
    <th> Produktname </th>
    <th> Euro-Preis </th>
    <th> Dollar-Preis </th>
  </tr>
  <tr>
    <td> Tolle Seife </td>
    <td> 2 Euro </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Dollar", 2));
?>
    </td>
  </tr>
  <tr>
    <td> Tolle Zahnpasta </td>
    <td> 3.5 Euro </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Dollar", 3.5));
?>
    </td>
  </tr>
  <tr>
    <td> Zahnseide </td>
```

```
    <td> 4.5 Euro </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Dollar", 4.5));
?>
     </td>
  </tr>
</table>
</body>
</html>
```

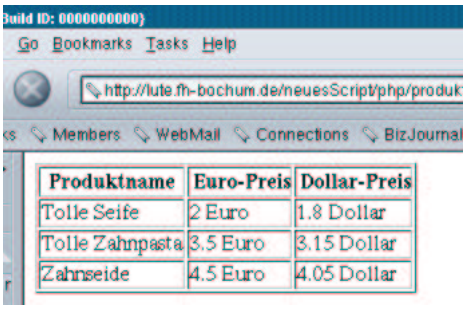


Abbildung 7.7: Ausgabe von Beispiel 7.10

Und hier sieht man den großen Vorteil der Programmierung mit Funktionen:
Wir haben die Funktion zur Euro-Dollar-Umrechnung einmal geschrieben. Benutzen können wir sie beliebig oft und, nachdem wir die Datei, die die Funktion enthält, eingebunden haben, auch aus beliebig vielen Seiten. Der Benutzer, der die Seiten erstellt, muss nicht einmal „Ahnung“ von php haben. Wir können ihm mitteilen, was er in die Seite einfügen muss, den Rest kann eigentlich jeder selber machen.

Auch eine Umstellung der Währungsbasis von Euro auf Dollar fällt nicht schwer:

Beispiel 7.11 *Produktseite: Umstellung der Währungsbasis*

```
<!-- Programm zur Anwendung der Euro-Dollar Umrechnung
  Dateiname: produkte2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Sanit&auml;rartikel</title>
<?php
    require_once("./includes/euroDollarUmrechnung.inc.php");
?>
</head>
<body>
<table border="1">
  <tr>
    <th> Produktname </th>
    <th> Euro-Preis </th>
    <th> Dollar-Preis </th>
  </tr>
```

```
  <tr>
    <td> Tolle Seife </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Euro", 2));
?>
    </td>
    <td>  2 Dollar</td>
  </tr>
  <tr>
    <td> Tolle Zahnpasta </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Euro", 3.5));
?>
    </td>
    <td> 3.5 Dollar </td>
  </tr>
  <tr>
    <td> Zahnseide </td>
    <td>
<?php
        echo(euroDollarUmrechnung("Euro", 4.5));
?>
     </td>

    <td> 4.5 Dollar </td>
  </tr>
</table>
</body>
</html>
```

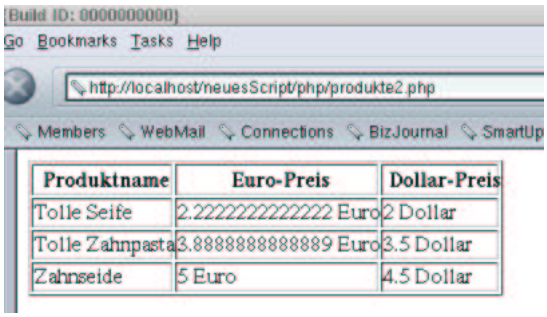


Abbildung 7.8: Ausgabe von Beispiel 7.11

Nun die Problemlösung in Javascript: Zunächst die Definition der Funktion:

Beispiel 7.12 *Euro zum Dritten in JavaScript: Die ausgelagerte Berechnungsdatei*

```
// Funktion zur Dollar-Euro oder Euro-Dollar Umrechnung
// Datei:euroDollarUmrechnung.js
// Verzeichnis: javascript
```

```
    function euroDollarUmrechnung(zielwaehrung, betrag)
    {
        var kurs=0.9;
        if((zielwaehrung=="Dollar")||(zielwaehrung=="dollar"))
        {
            dollarbetrag=kurs*betrag;
            return (dollarbetrag + " Dollar");
        }
        if((zielwaehrung=="Euro")||(zielwaehrung=="euro"))
        {
            eurobetrag=(1/kurs)*betrag;
            return  (eurobetrag + " Euro") ;
        }
    }
```

Dann der Aufruf aus einer html-Seite:

Beispiel 7.13 *Euro zum Dritten in JavaScript: Der Aufruf*

```
<!-- Euro-Dm Umrechnung Teil 3
  Dateiname: euro3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 3</title>
  <script language = "JavaScript"
            src="./javascript/euroDollarUmrechnung.js">
  </script>
</head>
<body>
    <script language = "JavaScript">
        var zielwaehrung;
        var betrag;
        var ergebnis;
        zielwaehrung=prompt("Bitte geben Sie die Zielwährung ein!","");
        if((zielwaehrung=="Dollar")||(zielwaehrung=="dollar") ||
            (zielwaehrung=="euro")||(zielwaehrung=="Euro"))
        {
            betrag=prompt("Bitte geben Sie den Betrag ein!","");
            ergebnis=euroDollarUmrechnung(zielwaehrung, betrag)
            document.write(ergebnis);
        }
        else
        {
            document.write("Falsche Zielw&auml;hrung: <br>" +
                                "Erlaubt sind: Euro oder Dollar!");
        }
    </script>
</body>
</html>
```

Auch hier wird die Funktion im <head>-Teil der html-Seite geladen. Im <body>-Teil wird sie aufgerufen. Der Code entspricht weitgehend Beispielen 7.7 und 7.8, so dass ich auf eine nähere Erläuterung verzichte. Auch an den Ein- und Ausgaben ändert sich nichts gegenüber Abb. 6.7, so dass ein weiterer Screenshot entfällt.

7.3.2 Volatilitäten (fortgesetzt)

Volatilität Bewertung kommt noch

7.3.3 Raketenbeispiel (fortgesetzt)

Zum Abschluss dieses Kapitels wollen wir das Raketenbeispiel erweitern. Um den Programmcode übersichtlicher zu halten, soll der Switch in eine eigene Funktion ausgelagert werden. Zunächst die ausgelagerten Funktionen.

Beispiel 7.14 *Raketen zum Vierten: Die ausgelagerte Berechnungsdatei*

```
<?php
// datumsfunktionen
// Datei:datumfunktionen.inc.php
// Verzeichnis: includes

    function tageInMonaten($monat, $tag)
    {
        switch($monat)
        {
            case 1:
                $tag=$tag; //ueberflussig, nur der Klarheit wegen
                break;
            case 2:
                $tag=31+$tag;
                break;
            case 3:
                $tag=31+28+$tag;
                break;
            case 4:
                $tag=31+28+31+$tag;
                break;
            case 5:
                $tag=31+28+31+30+$tag;
                break;
            case 6:
                $tag=31+28+31+30+31+$tag;
                break;
            case 7:
                $tag=31+28+31+30+31+30+$tag;
                break;
            case 8:
                $tag=31+28+31+30+31+30+31+$tag;
                break;
            case 9:
                $tag=31+28+31+30+31+30+31+31+$tag;
                break;
```

```
            case 10:
                $tag=31+28+31+30+31+30+31+31+30+$tag;
                break;
            case 11:
                $tag=31+28+31+30+31+30+31+31+30+31+$tag;
                break;
            case 12:
                $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
                break;
        }
        return $tag;
    }

    function berechneSekunden($tag, $stunden, $minuten, $sekunden)
    {
        return($tag*24*3600+
                    $stunden*3600+$minuten*60+$sekunden);
    }
?>
```

Nun das Hauptprogramm:

Beispiel 7.15 *Raketen zum Vierten: Der Aufruf*

```
<!-- raketenbeispiel 4 des Textes
  Dateiname: raketen4.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
<?php
   //Funktionen einlesen
   require_once("./includes/datumfunktionen.inc.php");
?>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br> <hr>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren

        echo "<form name='raketen2' action='$PHP_SELF' method='post'>";
?>
        <table border>
            <tr>
                <td>
                    Startmonat
```

```
    </td>
    <td>
        <input type="text" name="startmonat" size=12>
    </td>
</tr>
<tr>
    <td>
        Starttag
    </td>
    <td>
        <input type="text" name="starttag" size=12>
    </td>
</tr>

<tr>
    <td>
        Startzeit Stunden
    </td>
    <td>
        <input type="text" name="startStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Minuten
    </td>
    <td>
        <input type="text" name="startMinuten" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Sekunden
    </td>
    <td>
        <input type="text" name="startSekunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landemonat
    </td>
    <td>
        <input type="text" name="landemonat" size=12>
    </td>
</tr>
<tr>
    <td>
        Landetag
    </td>
    <td>
        <input type="text" name="landetag" size=12>
    </td>
</tr>
```

```
            <tr>
                <td>
                    Landezeit Stunden
                </td>
                <td>
                    <input type="text" name="landeStunden" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landezeit Minuten
                </td>
                <td>
                    <input type="text" name="landeMinuten" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landezeit Sekunden
                </td>
                <td>
                    <input type="text" name="landeSekunden" size=12>
                </td>
            </tr>
            <tr>
                <td colspan="2" align="center">
                    <input type="submit" name="Button1" value="Abschicken">
                </td>
            </tr>
        </table>
        </form>
<?php
    }
    else
    {
        // zweiter Aufruf nun rechnen
        // zunaechst starttage ausrechnen
        $starttag=tageInMonaten($startmonat,$starttag);
        // nun landetage ausrechnen
        $landetag=tageInMonaten($landemonat,$landetag);
        $startzeitInSekunden=berechneSekunden($starttag,
                    $startStunden,$startMinuten,$startSekunden);
        $landezeitInSekunden=berechneSekunden($landetag,
                    $landeStunden,$landeMinuten,$landeSekunden);
        // flugzeitInSekunden berechnen
        $flugzeitInSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitInSekunden<0)
        {
            echo "Fehleingabe: Landezeit vor Startzeit!";
        }
        else
        {
            //Flugzeit umrechnen, zuerst Sekunden und Minuten
```

```
            $flugzeitMinuten=floor($flugzeitInSekunden/60);
            $flugzeitSekunden=$flugzeitInSekunden%60;
            //nun minuten und stunden
            $flugzeitStunden=floor($flugzeitMinuten/60);
            $flugzeitMinuten=$flugzeitMinuten%60;
            $flugzeitTage=floor($flugzeitStunden/24);
            $flugzeitStunden=$flugzeitStunden%24;
            //ausgeben
            if($flugzeitTage==0)
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
            else
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitTage Tage <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
        }
    }
?>
</body>
</html>
```

Der Code ist bei der Verwendung von Funktionen klarer und kompakter. Durch die Auslagerung des Switches haben wir den Source-Code reduziert. Darüber hinaus steht der Algorithmus der Umrechnung in einer eigenen Funktion zur Verfügung. Dies ist auch der Grund für die Funktion berechneSekunden. Schreibersparnis haben wir dadurch nicht. Jeder aber, der aus Tagen, Stunden, Minuten und Sekunden, Sekunden berechnen will, kann den Code in der Funktion nutzen, ohne sich zu überlegen, wie das eigentlich geht.

Sie werden sich jetzt sicherlich [17] fragen, warum wir das Umgekehrte, also die Berechnung von Tagen, Stunden, Minuten und Sekunden aus den Sekunden, also die Codezeilen

```
//Flugzeit umrechnen, zuerst Sekunden und Minuten
$flugzeitMinuten=floor($flugzeitInSekunden/60);
$flugzeitSekunden=$flugzeitInSekunden%60;
//nun minuten und stunden
$flugzeitStunden=floor($flugzeitMinuten/60);
$flugzeitMinuten=$flugzeitMinuten%60;
$flugzeitTage=floor($flugzeitStunden/24);
$flugzeitStunden=$flugzeitStunden%24;
```

des Hauptprogramms nicht in eine Funktion ausgelagert haben, obwohl der Algorithmus hier doch wesentlich komplexer ist. Der Grund liegt aber auf der Hand: Alle unseren bisherigen Programme haben genau einen Wert zurückgegeben, mehr war nicht und mehr können wir auch nicht. Hier müssten

[17] oder auch nur vielleicht, oder aber auch gar nicht :-)

wir aber vier Werte zurückgeben (Tage, Stunden, Minuten und Sekunden) und das können wir (noch) nicht.

Zum Abschluss jetzt die JavaScript-Lösung:

Beispiel 7.16 *Raketen zum Vierten (JavaScript): Die ausgelagerte Berechnungsdatei*

```
// datumsfunktionen
// Datei:datumfunktionen.js
// Verzeichnis: javascript
    function tageInMonaten(monat, tag)
    {
        switch(monat)
        {
            case 1:
                tag=tag; //ueberflussig, nur der Klarheit weegen
                break;
            case 2:
                tag=31+tag;
                break;
            case 3:
                tag=31+28+tag;
                break;
            case 4:
                tag=31+28+31+tag;
                break;
            case 5:
                tag=31+28+31+30+tag;
                break;
            case 6:
                tag=31+28+31+30+31+tag;
                break;
            case 7:
                tag=31+28+31+30+31+30+tag;
                break;
            case 8:
                tag=31+28+31+30+31+30+31+tag;
                break;
            case 9:
                tag=31+28+31+30+31+30+31+31+tag;
                break;
            case 10:
                tag=31+28+31+30+31+30+31+31+30+tag;
                break;
            case 11:
                tag=31+28+31+30+31+30+31+31+30+31+tag;
                break;
            case 12:
                tag=31+28+31+30+31+30+31+31+30+31+30+tag;
                break;
        }
        return tag;
    }

    function berechneSekunden(tag, stunden, minuten, sekunden)
```

```
    {
        return(tag*24*3600+
                    stunden*3600+minuten*60+sekunden);
    }
```

Nun das Hauptprogramm:

Beispiel 7.17 *Raketen zum Vierten (JavaScript): Der Aufruf*

```
<-- Raketenbeispiel 4 des Textes
  Dateiname: raketen4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
  <script language = "JavaScript"
            src="./javascript/datumfunktionen.js">
  </script>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und sodann die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br>
    <script language = "JavaScript">
        var startSekunden;
        var startMinuten;
        var startStunden;
        var starttag;
        var startmonat;
        var startzeitInSekunden;
        var landeStunden;
        var landeMinuten;
        var landeSekunden;
        var landetag;
        var landemonat;
        var landezeitInSekunden;
        var flugzeitInSekunden;
        var flugzeitStunden;
        var flugzeitMinuten;
        var flugzeitSekunden;
        var flugzeitTage;
        //Einlesen
        startmonat=prompt("Bitte geben Sie den Startmonat ein","");
        starttag=prompt("Bitte geben Sie den Starttag ein","");
        startStunden=prompt("Bitte geben Sie die Stunden der Startzeit ein","");
        startMinuten=prompt("Bitte geben Sie die Minuten der Startzeit ein","");
        startSekunden=prompt("Bitte geben Sie die Sekunden der Startzeit ein","");
        landemonat=prompt("Bitte geben Sie den Landemonat ein","");
        landetag=prompt("Bitte geben Sie den Landetag ein","");
        landeStunden=prompt("Bitte geben Sie die Stunden der " +
                            "Landezeit ein","");
        landeMinuten=prompt("Bitte geben Sie die Minuten der " +
```

```
                        "Landezeit ein","");
landeSekunden=prompt("Bitte geben Sie die Sekunden der " +
                        "Lande ein","");

//Umwandeln
startmonat=parseInt(startmonat);
starttag=parseInt(starttag);
startStunden=parseInt(startStunden);
startMinuten=parseInt(startMinuten);
startSekunden=parseInt(startSekunden);
landemonat=parseInt(landemonat);
landetag=parseInt(landetag);
landeStunden=parseInt(landeStunden);
landeMinuten=parseInt(landeMinuten);
landeSekunden=parseInt(landeSekunden);
// zunaechst starttage ausrechnen
starttag=tageInMonaten(startmonat,starttag);
// nun landetage ausrechnen
landetag=tageInMonaten(landemonat,landetag);
// jetzt in Sekunden umrechnen
startzeitInSekunden=berechneSekunden(starttag,
            startStunden,startMinuten,startSekunden);
landezeitInSekunden=berechneSekunden(landetag,
            landeStunden,landeMinuten,landeSekunden);
// flugzeitInSekunden berechnen
flugzeitInSekunden=landezeitInSekunden-startzeitInSekunden;
if(flugzeitInSekunden<0)
{
    document.write("Fehleingabe: Landezeit vor Startzeit!");
}
else
{
    //Flugzeit umrechnen, zuerst Sekunden und Minuten
    flugzeitMinuten=Math.floor(flugzeitInSekunden/60);
    flugzeitSekunden=flugzeitInSekunden%60;
    //nun minuten und stunden
    flugzeitStunden=Math.floor(flugzeitMinuten/60);
    flugzeitMinuten=flugzeitMinuten%60;
    flugzeitTage=Math.floor(flugzeitStunden/24);
    flugzeitStunden=flugzeitStunden%24;
    //ausgeben
    if(flugzeitTage==0)
    {
        document.write("Die Flugzeit betr&auml;gt: <br>" +
                        flugzeitStunden + " Stunden <br>" +
                        flugzeitMinuten + " Minuten <br>" +
                        flugzeitSekunden + " Sekunden <br>");
    }
    else
    {
        document.write("Die Flugzeit betr&auml;gt: <br>" +
                        flugzeitTage + " Tage <br>" +
                        flugzeitStunden + " Stunden <br>" +
                        flugzeitMinuten + " Minuten <br>" +
```

```
                                    flugzeitSekunden + " Sekunden <br>");
            }
        }
    </script>
</body>
</html>
```

Auf beispielhafte Ein- und Ausgaben der Programme verzichte ich. An der Logik hat sich nichts geändert. Unsere Programme sind „nur“ übersichlicher und damit leichter wartbar geworden. Darüber hinaus kann ein Teil der Funktionalität (nämlich die, die wir in Funktionen ausgelagert haben) auch von anderen Entwicklern genutzt werden.

7.4 Referenz- und Wertparameter

Programmiersprachen kennen normalerweise zwei Mechanismen der Variablenübergabe. Beim ersten Mechanismus erhält die Funktion das Recht, die Aktualparameter zu ändern. Dies ist zum Beispiel beim Einlesen von Daten notwendig. Diese Art der Variablenübergabe heißt “call by reference”, die so übergebenen Variablen Referenzparameter.

Beim zweiten Mechanismus kann die aufgerufene Funktion die übergebenen Variablen nicht ändern. Die Funktion kann in diesem Fall die übergebenen Variablen zwar scheinbar ändern, die Änderungen werden dem aufrufenden Programm aber nicht übermittelt. Diese Art der Variablenübergabe heißt “call by value”, die so übergebenen Variablen Wertparameter.

Wir machen uns auch dies sofort an Beispielen klar:

Beispiel 7.18 *Variablenänderung bei der Übergabe*

```
<!-- Das Programm zur Variablenübergabe
  Dateiname: variablenAenderung.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Variablen</title>
</head>
<body>
<?php
    function variablenAenderung($a)
    {
        $a=2000;
    }
    $a=9;
    variablenAenderung($a);
    $b=21;
    variablenAenderung($b);
    echo ("Der Wert von \$a: $a <br>");
    echo ("Der Wert von \$b: $b <br>");
?>
</body>
</html>
```

Beispiel 7.18 erzeugt folgende, in Abb. 7.9 dargestellte Ausgabe:

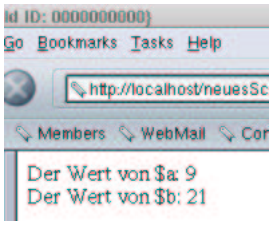


Abbildung 7.9: Ausgabe von Beispiel 7.18

Diese Ausgabe erscheint doch etwas merkwürdig. Schließlich haben wir zunächst eine Funktion deklariert, die den Wert ihres Übergabeparameters auf 2000 setzt:

```
 function variablenAenderung($a)
{
    $a=2000;
}
```

Dann haben wir die Funktion zweimal aufgerufen und stellen fest: Die Funktion macht gar nichts. Wir haben der Funktion Variable übergeben, die die Werte 9 resp. 21 hatten. Diese Werte haben die Variablen nach Durchlauf der Funktion immer noch.

Implizit haben sie übrigens gelernt, wie man das Dollarzeichen in php ausgibt. Das macht man, indem man dem Dollarzeichen den Backslash voranstellt. Der Backslash hebt die Sonderbedeutung des Dollarzeichens (und auch jedes anderen Zeichens, das eine Sonderbedeutung hat) auf. Er wird selbst nicht ausgegeben[18]. Ein Zeichen, wie den Backslash in php, gibt es in jeder Programmiersprache. Es heißt “Escape-Character”. Der “Escape-Character” in JavaScript ist übrigens auch der Backslash.

Wir können dieselbe Varhaltensweise bei der Variablenübergabe auch in JavaScript beobachten:

Beispiel 7.19 *Variablenänderung bei der Übergabe (JavaScript)*

```
<!-- Das Programm zur Variablenübergabe
  Dateiname: variablenAenderung.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Variablen</title>
</head>
<body>
<script language="javaScript">
    function variablenAenderung(a)
    {
        a=2000;
    }
    a=9;
    variablenAenderung(a);
    b=21;
    variablenAenderung(b);
    document.write("Der Wert von a: " + a + "<br>");
    document.write("Der Wert von b: " + b + "<br>");
```

[18] Was uns nun zu der Frage bringt: Wie wird wohl der Backslash ausgegeben?

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

Die Ausgabe ist ebenfalls in Abb. 7.9 dargestellt. Wie kommt sowas zustande? Die Erklärung ist einfach: Sowohl JavaScript als auch php benutzen bei „normalen“[19] Variablen Wertübergabe. Dies bedeutet, die Funktion kopiert den Wert der übergebenen Variable in einen anderen Speicherplatz. Mit der Kopie wird gerechnet. Wenn die Funktion sich beendet, findet umgekehrt kein Zurückkopieren statt. Alle an der Variable vorgenommenen Änderungen gehen verloren.

In JavaScript ist dies auch nicht änderbar. „Normale“ Variablen werden immer als Werte übergeben. In php können wir das ändern. Wie das geht, zeigt das nächste Beispiel:

Beispiel 7.20 *Variablenänderung bei der Übergabe (2)*

```
<!-- Das Programm zur Variablenübergabe
  Dateiname: variablenAenderung2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Variablen</title>
</head>
<body>
<?php
    function variablenAenderung(&$a)
    {
        $a=2000;
    }
    $a=9;
    variablenAenderung($a);
    $b=21;
    variablenAenderung($b);
    echo ("Der Wert von \$a: $a <br>");
    echo ("Der Wert von \$b: $b <br>");
?>
</body>
</html>
```

Beispiel 7.20 erzeugt die in Abb. 7.10 dargestellte Ausgabe:

Abb. 7.10 zeigt nun das gewünschte Verhalten. Die Veränderung der Variablen ist auch im Hauptprogramm sichtbar. Der einzige Unterschied zu Beispiel 7.18 findet sich in der Deklaration der Funktion:

```
function variablenAenderung(&$a)
```

Hier wird den Variablennamen des Übergabeparameters ($a) ein & vorangestellt. Dies sagt php: Sorge dafür, dass Änderungen dieser Variablen in der Funktion auch im Hauptprogramm sichtbar sind. php realisiert dies durch “call by reference”. Dies bedeutet einfach, dass die Funktion keine Kopie

[19] Was andere als normale Variable sind, erkläre ich später!

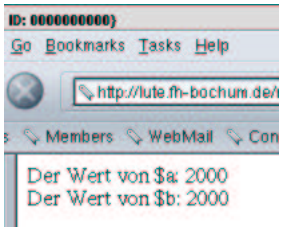


Abbildung 7.10: Ausgabe von Beispiel 7.20

der Übergabevariablen erhält, sondern einfach die Adresse im Hauptspeicher des Rechners, auf dem die Variable des Hauptprogramms abgelegt ist, mitgeteilt bekommt. Damit schreibt die Funktion bei Änderungen des Wertes der Variablen direkt in den auch dem Hauptprogramm zugänglichen Hauptspeicher. Damit bleiben Änderungen auch nach Beendigung der Funktion erhalten.

In php gilt also Folgendes:

- Schreiben wir nur den Namen einer Variablen in die Liste der Übergabeparameter, kann diese Variable in der Funktion nicht geändert werden.
- Stellen wir dem Namen einer Variablen in der Parameterliste ein & voran, wirken sich in der Funktion durchgeführte Änderungen an dieser Variablen auch im Hauptprogramm aus.

In php kennen Sie jetzt also zwei Wege, Informationen aus einer Funktion in das aufrufende Programm zu transportieren:

- Durch das `return`-Kommando.
- Durch “call by reference”.

In JavaScript sieht die Sache ein wenig anders aus. Variablen, die Zahlen, Strings oder Wahrheitswerte enthalten, werden immer “by value” übergeben[20]. Variablen, die Arrays oder Objekte[21] enthalten, werden als Referenzen übergeben, also durch JavaScript-Funktionen geändert. Dieses Verhalten ist auch nicht änderbar. In JavaScript kennen Sie also zunächst nur eine Möglichkeit, Informationen aus einer Funktion in das aufrufende Programm zu transportieren: Durch das `return`-Statement.

In der php-Lösung können wir nun unsere Datumsfunktionen erweitern, indem wir den Algorithmus, der Tage, Stunden, Minuten und Sekunden aus Sekunden extrahiert, ebenfalls in eine eigene Funktion auslagern:

Beispiel 7.21 *Raketen zum Fünften: Die ausgelagerte Berechnungsdatei*

```
<?php
// datumsfunktionen
// Datei:datumfunktionen2.inc.php
// Verzeichnis: includes

    function tageInMonaten($monat, $tag)
    {
        switch($monat)
        {
```

[20] Dies sind die oben angesprochenen „normalen“ Variablen.
[21] Kennen Sie beides noch nicht, doch keine Angst (oder gerade), sie werden sie noch kennenlernen.

```
            case 1:
                $tag=$tag; //ueberflussig, nur der Klarheit weegen
                break;
            case 2:
                $tag=31+$tag;
                break;
            case 3:
                $tag=31+28+$tag;
                break;
            case 4:
                $tag=31+28+31+$tag;
                break;
            case 5:
                $tag=31+28+31+30+$tag;
                break;
            case 6:
                $tag=31+28+31+30+31+$tag;
                break;
            case 7:
                $tag=31+28+31+30+31+30+$tag;
                break;
            case 8:
                $tag=31+28+31+30+31+30+31+$tag;
                break;
            case 9:
                $tag=31+28+31+30+31+30+31+31+$tag;
                break;
            case 10:
                $tag=31+28+31+30+31+30+31+31+30+$tag;
                break;
            case 11:
                $tag=31+28+31+30+31+30+31+31+30+31+$tag;
                break;
            case 12:
                $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
                break;
        }
        return $tag;
    }

    function berechneSekunden($tag, $stunden, $minuten, $sekunden)
    {
        return($tag*24*3600+
                    $stunden*3600+$minuten*60+$sekunden);
    }

    function tageStundenMinutenSekundenAusSekunden(&$tage,
                        &$stunden, &$minuten, &$sekunden)
    {
        // zuerst sekunden
        $minuten=floor($sekunden/60);
        $sekunden=$sekunden%60;
        //nun minuten und stunden
        $stunden=floor($minuten/60);
```

```
        $minuten=$minuten%60;
        //und tage
        $tage=floor($stunden/24);
        $stunden=$stunden%24;
    }
?>
```

Nun das Hauptprogramm:

Beispiel 7.22 *Raketen zum Fünften: Der Aufruf*

```
<!-- raketenbeispiel 5 des Textes
  Dateiname: raketen5.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
<?php
   //Funktionen einlesen
   require_once("./includes/datumfunktionen2.inc.php");
?>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br> <hr>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren

        echo "<form name='raketen2' action='$PHP_SELF' method='post'>";
?>
        <table border>
            <tr>
                <td>
                    Startmonat
                </td>
                <td>
                    <input type="text" name="startmonat" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Starttag
                </td>
                <td>
                    <input type="text" name="starttag" size=12>
                </td>
            </tr>
```

```
<tr>
    <td>
        Startzeit Stunden
    </td>
    <td>
        <input type="text" name="startStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Minuten
    </td>
    <td>
        <input type="text" name="startMinuten" size=12>
    </td>
</tr>
<tr>
    <td>
        Startzeit Sekunden
    </td>
    <td>
        <input type="text" name="startSekunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landemonat
    </td>
    <td>
        <input type="text" name="landemonat" size=12>
    </td>
</tr>
<tr>
    <td>
        Landetag
    </td>
    <td>
        <input type="text" name="landetag" size=12>
    </td>
</tr>

<tr>
    <td>
        Landezeit Stunden
    </td>
    <td>
        <input type="text" name="landeStunden" size=12>
    </td>
</tr>
<tr>
    <td>
        Landezeit Minuten
    </td>
```

```
                <td>
                    <input type="text" name="landeMinuten" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landezeit Sekunden
                </td>
                <td>
                    <input type="text" name="landeSekunden" size=12>
                </td>
            </tr>
            <tr>
                <td colspan="2" align="center">
                    <input type="submit" name="Button1" value="Abschicken">
                </td>
            </tr>
        </table>
        </form>
<?php
    }
    else
    {
        // zweiter Aufruf nun rechnen
        // zunaechst starttage ausrechnen
        $starttag=tageInMonaten($startmonat,$starttag);
        // nun landetage ausrechnen
        $landetag=tageInMonaten($landemonat,$landetag);
        $startzeitInSekunden=berechneSekunden($starttag,
                $startStunden,$startMinuten,$startSekunden);
        $landezeitInSekunden=berechneSekunden($landetag,
                $landeStunden,$landeMinuten,$landeSekunden);
        // flugzeitInSekunden berechnen
        $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitInSekunden<0)
        {
            echo "Fehleingabe: Landezeit vor Startzeit!";
        }
        else
        {
            //Flugzeit umrechnen
            tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                            $flugzeitStunden, $flugzeitMinuten,
                            $flugzeitSekunden);
            //ausgeben
            if($flugzeitTage==0)
            {
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
            else
            {
```

```
                echo ("Die Flugzeit betr&auml;gt: <br>" .
                                "$flugzeitTage Tage <br>" .
                                "$flugzeitStunden Stunden <br>" .
                                "$flugzeitMinuten Minuten <br>" .
                                "$flugzeitSekunden Sekunden <br>");
            }
        }
    }
?>
</body>
</html>
```

Auch hier sehen wir noch einmal: Das & wird den Variablennamen bei der Deklaration der Funktion vorangestellt, der Aufruf bleibt wie gehabt.

Aufgabe 7.1 *Sie sollen für eine Bank die Errechnung von Darlehenskonditionen für Kunden der Bank über das Internet ermöglichen. Eingegeben werden soll das Eigenkapital und der Preis der Immobilie, die gekauft werden soll. Der Zinssatz ist 5 %, die Tilgung 1 %. Das Programm soll die monatliche Belastung ausgeben. Wenn die Eigenkapitalquote des Kunden kleiner als 30 % ist, soll keine Berechnung durchgeführt werden und anstelle dessen ausgegeben werden, dass die Bank Immobilienerwerb mit einer so geringen Eigenkapitalquote nicht finanziert.*

Berechnung der Eigenkapitalquote und Berechnung der monatlichen Belastung sollen in eigene Funktionen ausgelagert werden!

Aufgabe 7.2 *Eine andere Bank vergibt Kredite für das Privatkundengeschäft nach folgenden Kriterien:*

- *Zunächst wird der Preis der Immobilie ermittelt (indem der Kunde ihn angibt).*
- *Dann wird das Eigenkapital ermittelt (indem der Kunde es angibt).*
- *Dann wird die Tilgung ermittelt (indem der Kunde sie angibt).*
- *Bei bereits bestehenden Immobilien entspricht der Wert der Immobilie dem Kaufpreis. Wenn es sich hingegen um einen Neubau handelt zieht die Bank 20% vom Kaufpreis ab, um den Wert der Immobilie zu ermitteln.*
- *Sodann gibt es 4 Zinsbereiche:*
  - *Für das aufzunehmende Geld bis zu 60% des Wertes der Immobile 6,25%.*
  - *Für das aufzunehmende Geld zwischen 60% und 80 % des Wertes der Immobile 7%.*
  - *Für das aufzunehmende Geld zwischen 80% und 100 % des Wertes der Immobile 7,5%.*
  - *Für das aufzunehmende Geld über 100% des Wertes der Immobile 8,5%. (das kann wg. des 20% Abzugs bei Neubauten passieren).*

Diese Bank hat keine Skrupel und verleiht ihr Geld auch, wenn der Kunde kein Eigenkapital hat (Hauptsache die Sicherheit stimmt).

Auch hier soll die Berechnung der monatlichen Belastung und die Berechnung des Wertes der Immobilie in Funktionen ausgelagert werden!

Kapitel 8

Klassen und Objekte

8.1 Motivation

Der Entwickler einer neuen Software steht vor der immer gleichen Herausforderung, einen kleinen Ausschnitt der Realität in einem Programm abzubilden. Das fertige Produkt soll Menschen bei der Erledigung bestimmter Arbeiten unterstützen oder sie sogar ganz übernehmen. Dazu müssen Dinge aus dem „wirklichen Leben“ und das, was man mit ihnen machen kann, in irgendeiner abstrakten Form im Computer durch ein Programm abgebildet werden.

Genau dies haben wir in unseren bisherigen Beispielen getan. In den Übungen haben Sie ein Programm geschrieben, um die Berechnung monatlicher Belastungen bei Baukrediten zu automatisieren. In der Vorlesung haben wir uns mit Währungsumrechnungen oder Flugdauerberechnungen beschäftigt. Betrachten wir das Kreditprogramm aus der Übung noch einmal aus einer anderen Perspektive: Eigentliches Ziel dort ist, Tätigkeit im Zusammenhang mit Krediten zu automatisieren. Hier können wir feststellen:

- Kredite haben Eigenschaften z.B.:
  - Die Kredithöhe
  - Die monatliche Belastung, die mit dem Kredit verbunden ist.
  - Der Zinssatz zu dem der Kredit vergeben wird.
  - Die vom Benutzer gewünschte Tilgung.
- Mit Krediten sind Tätigkeiten verbunden:
  - Die Kredithöhe kann berechnet werden (aus Eigenkapital und Immobilienpreis).
  - Die monatliche Belastung kann berechnet werden (aus Kredithöhe, Zins und Tilgung).

Wenn wir unsere Flugdauerberechnung mal ganz genau anschauen, können wir ähnliches feststellen:

- Auch dort gibt es Eigenschaften:
  - Die Startzeit.
  - Die Landezeit.
  - Die Flugzeit.
- Auch dort gibt es Tätigkeiten:
  - Wir müssen Zeiten in Sekunden umrechnen.

- Der umgekehrte Vorgang musste auch programmiert werden.
- Die Flugzeit insgesamt musste berechnet werden.

Mit einigem Nachdenken erkennen wir auch Zusammenhänge mit unseren Programmierkonstrukten:

- Für die Eigenschaften sind Variablen oder Konstante zuständig. Für Kredithöhe, Immobilienpreis, Zinssätze, Start- oder Landezeiten hatten wir jeweils Variablen oder Konstante eingeführt.
- Die Tätigkeiten sind der Programmablauf, den Sie, wie Sie in Kapitel 7 gelernt haben, in Funktionen auslagern können.

Prozedurale Entwicklung

Die Vorgehensweise, die wir bisher angewendet haben, heißt prozedurale Entwicklung. Und das bedeutet: Wir entwickeln den Algorithmus (heißt wir beschreiben die funktionalen Abläufe), ermitteln die benötigten Variablen und schreiben dann das Programm. Zwischen Variablen und Funktionen besteht kein Zusammenhang, außer dem, dass die Funktionen die Variablen benutzen. Wir haben also eine Trennung von Daten (dies sind ja die Variablen) und Funktionen.

Für unsere Zwecke ist dies auch völlig ausreichend. Für kleinere Anwendungen, wie unsere bisherigen Beispiele, ist der prozedurale Ansatz völlig okay und Objektorientierung schlicht Overkill[1].

Allerdings sind unsere bisherigen Beispiele auch sehr begrenzt. Denn wenn wir unser Kreditbeispiel betrachten, so besteht der Ausschnitt der Wirklichkeit, den wir modellieren und implementieren eigentlich nur aus Krediten. Alle Variablen beziehen sich auf Kredite und alle Funktionen machen irgendetwas im Zusammenhang mit der Kreditberechnung. Bei den anderen Beispielen ist dies nicht viel anders, wie eine kurze Betrachtung der Flugdauerberechnung oder der Euro-Dollar-Umrechnung zeigt.

Schon die Betrachtung eines Handelsunternehmens, dass seine Lieferanten- und Kundenbeziehung über Software[2] abbildet, zeigt ungleich kompliziertere Zusammenhänge.

Denn hier haben wir plötzlich Lieferanten, Kunden, Aufträge, Angebote, Lieferungen, Rechnungen und was der Dinge mehr sind. Wir haben demzufolge später in der Anwendung Variable für Kundendaten, für Lieferantendaten, für Rechnungen von Lieferanten, Rechnungen an Kunden usw. Darüberhinaus werden wir Funktionen benötigen, um diese Variablen zu manipulieren. Hier kommt nun die Objektorientierung ins Spiel.

Objektorientierung

Die Theorie der Objektorientierung hebt genau die die prozedurale Programmierung kennzeichnende Trennung von Daten und Funktionen auf und fasst beide zu einer Einheit zusammen. Mit dieser Einheit wird ein Objekt aus der realen Welt, der so genannte „Problembereich“, beschrieben.

Tatsächlich ist ein Objekt eine Abstraktion eines Gegenstandes oder Wesens der realen Welt, z.B. eines Kunden, eines Lieferanten oder eines Auftrags. Hierbei bedeutet „Abstraktion“, sich auf das Wesentliche für das zu lösende Problem zu konzentrieren und nicht relevante Details weg zu lassen. In ein objektorientiertes Modell muss man die für die betrachtete Problemstellung erforderlichen Aspekte übernehmen. Die jeweils sinnvolle und hilfreiche Abstraktion zu erkennen ist gerade das zentrale Problem der objektorientierten Softwareentwicklung.

[1] Auch wenn das Anhänger der strikten Objektorientierung anders sehen.

[2] Was diese Unternehmen ja alle machen.

Hierzu ein kurzes Zwischenbeispiel: Wenn Sie eine Software für Online-Autorennen entwickeln wollen, so sind Autos (hier Rennwagen) sicher Gegenstände, die Sie modellieren müssen. Diese Autos haben Eigenschaften, wie Farbe[3], PS, gegenwärtige Geschwindigkeit, gegenwärtige Position auf der Rennstrecke und Funktionen wie beschleunigen und abbremsen.

Wenn Sie hingegen Software für einen Autoverleih schreiben müssen, sind diese Dinge nicht so wirklich von Interesse, die Farbe spielt meist keine Rolle, die gegenwärtige Geschwindigkeit oder Position auf der Rennstrecke gibt es nicht. Funktionen zum Beschleunigen oder Abbremsen eines Wagens müssen Sie ganz sicherlich nicht schreiben. Dafür interessiert plötzlich der Preis des Wagens, zu welcher Klasse[4] er gehört, wie viele Sie wo haben usw. Und die Funktionen sind jetzt eher dafür zuständig, Wagen zu verleihen oder ihn wieder zurück zu nehmen.

Sie sehen also, dass sich für unterschiedliche Anwendungen auch unterschiedliche Abstraktionen ergeben.

Die Hauptidee ist, dass ein Objekt allein durch seine Daten (Eigenschaften) und seine Funktionen (Verhalten) bestimmt ist. In der Theorie der Objektorientierung nennt man die Daten auch Attribute, Instanzvariablen oder Eigenschaften. In der JavaScript-Dokumentation wird von Eigenschaften (engl. properties) geprochen. Auch die Funktionen ändern ihren Namen, sie heißen jetzt Methoden[5].

Klassen

Kehren wir kurz zum Rennwagenbeispiel zurück. Nehmen wir an, wir wollen tatsächlich ein Formel 1-Spiel entwickeln. dann müssen wir (Stand 2001) 22 Rennwagen modellieren. Die sind aber irgendwie alle gleich. Das heißt, alle haben eine Farbe, PS, gegenwärtige Geschwindigkeit, gegenwärtige Position auf der Rennstrecke und Funktionen wie beschleunigen und abbremsen. Die Autos haben zwar verschiedene Farben, unterschiedliche PS und auch unterschiedliche gegenwärtige Geschwindigkeiten und gegenwärtige Positionen, aber immerhin, alles dieses haben diese Rennwagen. Wir können sagen, diese Eigenschaften kennzeichnen das Bild eines Rennwagens in unserer Abstraktion. Abb. 8.1 fasst dies noch einmal zusammen:

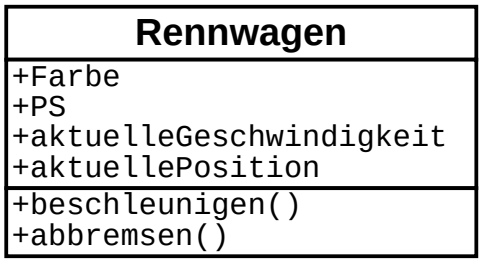


Abbildung 8.1: Ein Rennwagen in unserer Abstraktion

Abb. 8.1 bedient sich einer genormten Notation, der UML (Unified Modelling Language). Hier werden die Dinge dargestellt, die einen Rennwagen in unserer Abstraktion kennzeichnen. Wir sehen zunächst den Namen (Rennwagen), dann im nächsten Rechteck die Instanzvariablen und im letzten Rechteck die Methoden.

Betrachten wir hingegen Michael Schumachers Auto zu einem bestimmten Zeitpunkt im Rennen. Dies könnte wie in Abb. 8.2 dargestellt aussehen:

[3] muss, schließlich soll Michael Schumachers Ferrari ja rot sein, der Williams/BMW von seinem Bruder jedoch besser nicht.

[4] gemeint ist: Kleinwagen, Mittelklasse, S-Klasse

[5] Ja, so sind sie, die Informatiker, erfinden ständig neue Namen.

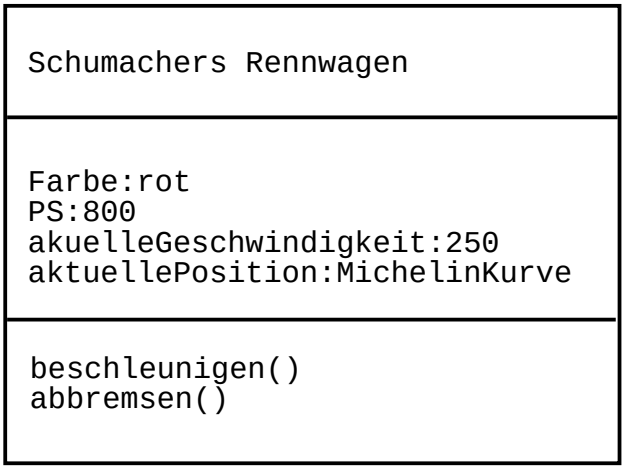


Abbildung 8.2: Schumachers Rennwagen

Abb. 8.1 und Abb. 8.2 unterscheiden sich eigentlich an nicht vielen Punkten: Abb. 8.1 zeigt, wie Rennwagen allgemein repräsentiert werden, Abb. 8.2 hingegen ein bestimmtes Auto. Dies zeigt sich daran, dass die Instanzvariablen konkrete Werte erhalten. Abb. 8.1 ist also die Vorschrift, nach der alle 22 Rennwagen (Objekte) des Formel-1-Spiels konstruiert werden.

Eine wie in Abb. 8.1 dargestellte Konstruktionsvorschrift nennen wir Klasse. Eine wie in Abb. 8.2 dargestellte Konkretisierung einer Klasse nennen wir ein Objekt der Klasse.

8.2 Beispiele in php

Veranschaulichen wir uns Klassen und Objekte an einem einfachen Beispiel. Wir wollen unser Euro-Dollar-Umrechnungsprogramm objektorientiert schreiben.

8.2.1 Euro-Dollar-Umrechnung objektorientiert

Bei der objektorientierten Entwickllung müssen wir uns zu Anfang etwas Gedanken machen:

Als erstes müssen wir uns überlegen, wieviele Klassen wir benötigen. Zunächst stellen wir fest, wir brauchen eine Klasse für die Euros.

Wir könnten auf die Idee kommen, auch noch eine Klasse für Dollars vorzusehen. Doch der Wert in Euro und der Wert in Dollar sind eigentlich zwei Seiten derselben Medaille. Wenn wir nämlich in einem Objekt einen Betrag in Euro abspeichern, können wir in jederzeit auch in Dollar (durch einfache Umrechnung) ausgeben.

Dasselbe könnte man sich auch in Bezug auf die Dollars überlegen. D.h. es ist eigentlich egal, ob wir eine Klasse für Euros oder eine für Dollars erzeugen, eine Klasse reicht. Der eigentliche Grund ist, dass die Klasse eigentlich Währung heißen müsste. Und in welcher Währung wir den Wert ausgeben, ist eigentlich egal.

Wir nennen unsere Klasse Euros. Weitere Klassen sind nicht notwendig. Als nächstes legen wir die Instanzvariablen fest. Das ist einfach, wir haben zwei, nämlich den Eurobetrag und den Kurs. Wir nennen die Variablen betrag bzw. kurs. Sodann muss es Methoden geben. Wir müssen zunächst den Eurobetrag eingeben und auch wieder ausgeben können. Hierfür benötigen wir zwei Methoden. Methoden die eine Instanzvariable setzen, nennen wir setNamenDerVariable. Unsere erste Methode

heißt also setBetrag. Analog heißen Methoden, die den Namen einer Instanzvariablen ausgeben getNamenDerVariable, hier also getBetrag. Wir müssen in Dollar umrechnen können, ergibt eine weitere Methode, wir nennen sie konvertiereInDollar und wir müssen aus Dollars Euros erzeugen können, ergibt eine vierte Methode, der wir den Namen importiereAusDollar geben.

Abb. 8.3 zeigt dies in UML-Notation.

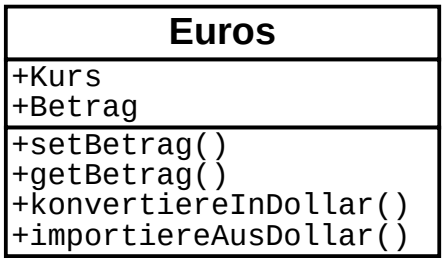


Abbildung 8.3: Die Euro-Klasse in UML

Die Arbeit mit Klassen ähnelt der Arbeit mit Funktionen. Zunächst wird die Klasse implementiert, danach können wir sie benutzen. Wir starten mit der Implementierung der Klasse. Ich werde zunächst den Code darstellen. Danach werde ich ihn erklären.

Beispiel 8.1 *Die Klasse für Euros*

```
<?php
    class Euro
    {
        var $betrag;
        var $kurs=0.9;

        function setBetrag($betrag)
        {
            $this->betrag=$betrag;
        }
        function getBetrag()
        {
            return $this->betrag;
        }
        function konvertiereInDollar()
        {
            return($this->betrag*$this->kurs);
        }
        function importiereAusDollar($betrag)
        {
            $this->betrag=$betrag*(1/$this->kurs);
        }
    }
?>
```

Klassen beginnen mit dem Schlüsselwort class. Die Implementierung der Klasse wird in geschweifte Klammern eingeschlossen:

```
class Euro
{

}
```

Danach sehen wir eine Variablendeklaration, wie wir sie bisher nur von JavaScript kennen. Instanzvariablen werden bei der Implementierung von Klassen, wie es ja in JavaScript immer möglich ist, durch das Schlüsselwort var deklariert:

```
var $betrag;
var $kurs=0.9;
```

Hierauf folgen die Methoden. Methoden entsprechen Funktionen und werden auch so implementiert. Die Implementierung erfolgt also durch das Schlüsselwort function gefolgt von den Übergabeparametern in runden Klammern. Die Anweisungen der Funktion werden, wie bekannt, in geschweifte Klammern eingeschlossen.

```
function setBetrag($betrag)
{
    $this->betrag=$betrag;
}
```

Neu ist die eigentümliche Konstruktion:

```
    $this->betrag=$betrag;
```

$this ist das Objekt der Klasse selber. In Prosa bedeutet oben dargestellte Zeile: Setze den Wert der Instanzvariable $betrag auf den Wert des Übergabeparameters $betrag. Hier haben zwei Variablen den gleichen Namen[6]. Dies ist möglich, weil auf die Instanzvariablen innerhalb der Klassenimplementierung immer mit der Syntax $this->NameDerVariable zugegriffen wird. Den Rest der Implementierung sollten Sie nun verstehen[7]. So bedeutet die Anweisung

```
    return($this->betrag*$this->kurs);
```

innerhalb der Methode konvertiereInDollar: Nimm den Inhalt der Instanzvariable $betrag ($this->betrag) und multipliziere dies mit dem Inhalt der Instanzvariable $kurs ($this->kurs) und gib das Ergebnis an das aufrufende Programm zurück.

Nun müssen wir uns noch angucken, wie unsere Methoden aufgerufen werden. Dies zeigt Beispiel 8.2.

Beispiel 8.2 *Aufruf der Euro-Klasse*

```
<!-- Programm zur Euro-Dollar Umrechnung Teil3
  Dateiname: euro5.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>
        Euro-Dollar Umrechnung Teil 5:
```

[6] Ich hätte dies einfacher gestalten können, indem ich der Übergabevariablen einen anderen Namen gegeben hätte, das wäre natürlich auch gegangen, aber in den meisten Code-Beispielen wird es so gemacht wie in meinem Beispiel und zum Anderen, wenn schwierig, dann auch richtig schwierig!

[7] Auch wenn Sie das noch nicht nutzen können :-).

```
        Obektorientiert
  </title>
<?php
    require_once("./klassen/Euro.inc.php");
?>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='euro2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Zielw&auml;hrung
                    </td>
                    <td>
                        <input type="text" name="zielwaehrung" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Betrag
                    </td>
                    <td>
                        <input type="text" name="betrag" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
            if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
            {
                $euro=new Euro();
                $euro->setBetrag($betrag);
                $dollars=$euro->konvertiereInDollar();
                echo "Ihre Eingabe entspricht $dollars! Dollar";

            }
            else
            {
                if(($zielwaehrung=="euro")||($zielwaehrung=="Euro"))
                {
```

```
                    $euro=new Euro();
                    $euro->importiereAusDollar($betrag);
                    $euros=$euro->getBetrag();
                    echo "Ihre Eingabe entspricht $euros! Euro";
                }
                else
                {
                    echo("Falsche Zielw&auml;hrung: <br>" .
                            "Erlaubt sind: Euro oder Dollar!");
                }
            }
    }
?>
</body>
</html>
```

Zunächst ist die Entscheidung, ob von Dollar in Euro oder umgekehrt umgerechnet werden soll, in die aufrufende php-Datei verlegt[8].

Objekte muss man sodann erzeugen. Dies geschieht z.B. in der Zeile:

```
$euro=new Euro();
```

Diese Zeile erzeugt ein Objekt vom Typ Euro und weist es der Variablen $euro zu. Variablen werden bei der objektorientierten Programmierung erweitert, sie können nun auch Objekte aufnehmen. Die Methoden der Objekte werden durch den Name des Objekts[9] gefolgt von `->` gefolgt vom Namen der Methode aufgerufen:

```
$euro->setBetrag($betrag);
```

Der Unterschied zum Funktionsaufruf besteht also nur darin, dass dem Namen der Methode der Namen des Objektes gefolgt von `->` vorangestellt wird.

Nun sollte der restliche Code aus Beispiel 8.2 verständlich sein. Die in Beispiel 8.1 implementierten Funktionen, die nun Methoden heißen, werden aufgerufen und das erzeugte Ergebnis ausgegeben.

Beispiele 8.2 und 8.1 sind natürlich komplizierter als die in Kapitel 7 dargestellte Lösung mit Funktionen. Solch eine einfache Aufgabe würde man auch nicht objektorientiert realisieren, sondern die Lösung aus Kapitel 7 vorziehen. Etwas anderes ist dies, wenn wir eine komplizierte Anwendung mit Funktionalitäten, Währungen betreffend realisieren müssten. Dann wäre der objektorientierte Ansatz vorzuziehen.

8.2.2 Volatilitäten (objektorientiert)

Volatilität Bewertung kommt noch

8.2.3 Raketenbeispiel (objektorientiert) und die Nutzung von Konstruktoren

Auch hier müssen wir uns erst einige Gedanken machen:
Die erste Überlegung ist immer, wieviele und welche Klassen benötigen wir. Auch hier versuchen wir es erst einmal mit einer Klasse für die Raketen.

[8] Erkennt man an den if-Anweisungen!

[9] Genauer der Name der Variablen, die das Objekt enthält, werden wir im folgenden aber immer mit Namen des Objekts abkürzen.

Zunächst müssen wir die Instanzvariablen ermitteln. Das ist aber einfach, es handelt sich um die Start- und Landezeiten. Als nächstes müssen die benötigten Methoden gefunden werden. Normalerweise müsste zunächst für jede Instanzvariable eine `set-` und eine `get-`Methode geschrieben werden.

Wenn wir wie im Euro-Beispiel (Kapitel 8.2.1) vorgehen würden, müssten wir in jedem, die Raketen-Klasse nutzenden Programm, ein Objekt vom Typ Rakete erzeugen (mit dem `new`-Kommando) und sodann alle Instanzvariablen mit `set-`Methoden setzen. Dies bauscht den Code auf, wir haben immerhin 10 Instanzvariablen. Für solche Fälle stellen objektorientierte Sprachen Konstruktoren zur Verfügung.

Konstruktoren sind besondere Methoden, die bei der Erzeugung von Objekten automatisch aufgerufen werden. Sie können u.a. dazu dienen, Vorbelegungen von Variablen (Initialisierungen) durchzuführen. Der Methodenname eines Konstruktors ist vorgegeben, er entspricht dem Klassennamen.

Neben dem Konstruktor benötigen wir eigentlich nur eine weitere Methode, nämlich die, die die Flugzeit berechnet. Abb. 8.4 zeigt dies zusasmmenfassend in einem UML-Klassendiagramm:

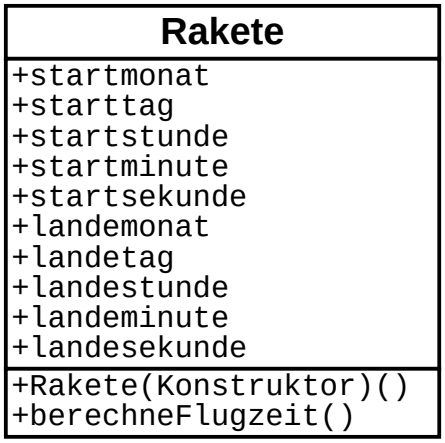


Abbildung 8.4: Die Raketen-Klasse in UML

Kommen wir nun zur Implementierung. Zunächst die Raketen-Klasse:

Beispiel 8.3 *Die Raketenklasse*

```
<?php
class Rakete
{
// Raketenklasse
// Datei:Rakete.inc.php
// Verzeichnis: klassen
    var $startmonat;
    var $starttag;
    var $startstunde;
    var $startminute;
    var $startsekunde;
    var $landemonat;
    var $landetag;
    var $landestunde;
    var $landeminute;
    var $landesekunde;
```

```
/*
*
*    Konstruktor
*
*/

function Rakete($startmonat, $starttag, $startstunde,
                $startminute, $startsekunde, $landemonat,
                $landetag, $landestunde, $landeminute,
                $landesekunde)
{
    $this->startmonat=$startmonat;
    $this->starttag=$starttag;
    $this->startstunde=$startstunde;
    $this->startminute=$startminute;
    $this->startsekunde=$startsekunde;
    $this->landemonat=$landemonat;
    $this->landetag=$landetag;
    $this->landestunde=$landestunde;
    $this->landeminute=$landeminute;
    $this->landesekunde=$landesekunde;
}

function berechneFlugzeit(&$flugzeitTage,
                        &$flugzeitStunden, &$flugzeitMinuten,
                        &$flugzeitSekunden)
{
    $starttag=$this->tageInMonaten($this->startmonat,$this->starttag);
    // nun landetage ausrechnen
    $landetag=$this->tageInMonaten($this->landemonat,$this->landetag);
    $startzeitInSekunden=$this->berechneSekunden
            ($starttag,$this->startstunde,$this->startminute,
                                                $this->startsekunde);
    $landezeitInSekunden=$this->berechneSekunden
            ($landetag,$this->landestunde,$this->landeminute,
                                                $this->landesekunde);
    // flugzeitInSekunden berechnen
    $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
    if($flugzeitSekunden<0)
    {
        return false;
    }
    $this->tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                        $flugzeitStunden, $flugzeitMinuten,
                        $flugzeitSekunden);
    return true;
 }

function tageInMonaten($monat, $tag)
{
    switch($monat)
    {
        case 1:
```

```
            $tag=$tag; //ueberflussig, nur der Klarheit weegen
            break;
        case 2:
            $tag=31+$tag;
            break;
        case 3:
            $tag=31+28+$tag;
            break;
        case 4:
            $tag=31+28+31+$tag;
            break;
        case 5:
            $tag=31+28+31+30+$tag;
            break;
        case 6:
            $tag=31+28+31+30+31+$tag;
            break;
        case 7:
            $tag=31+28+31+30+31+30+$tag;
            break;
        case 8:
            $tag=31+28+31+30+31+30+31+$tag;
            break;
        case 9:
            $tag=31+28+31+30+31+30+31+31+$tag;
            break;
        case 10:
            $tag=31+28+31+30+31+30+31+31+30+$tag;
            break;
        case 11:
            $tag=31+28+31+30+31+30+31+31+30+31+$tag;
            break;
        case 12:
            $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
            break;
    }
    return $tag;
}

function berechneSekunden($tag, $stunden, $minuten, $sekunden)
{
    return($tag*24*3600+
                $stunden*3600+$minuten*60+$sekunden);
}

function tageStundenMinutenSekundenAusSekunden(&$tage,
                    &$stunden, &$minuten, &$sekunden)
{
    // zuerst sekunden
    $minuten=floor($sekunden/60);
    $sekunden=$sekunden%60;
    //nun minuten und stunden
    $stunden=floor($minuten/60);
    $minuten=$minuten%60;
```

```
        //und tage
        $tage=floor($stunden/24);
        $stunden=$stunden%24;
    }
}
?>
```

Auch hier werden zunächst die Instanzvariablen deklariert:

```
var $startmonat;
var $starttag;
var $startstunde;
var $startminute;
var $startsekunde;
var $landemonat;
var $landetag;
var $landestunde;
var $landeminute;
var $landesekunde;
```

Dann folgt der Konstruktor:

```
function Rakete($startmonat, $starttag, $startstunde,
                $startminute,$startsekunde, $landemonat,
                $landetag, $landestunde, $landeminute,
                $landesekunde)
{
    $this->startmonat=$startmonat;
    $this->starttag=$starttag;
    $this->startstunde=$startstunde;
    $this->startminute=$startminute;
    $this->startsekunde=$startsekunde;
    $this->landemonat=$landemonat;
    $this->landetag=$landetag;
    $this->landestunde=$landestunde;
    $this->landeminute=$landeminute;
    $this->landesekunde=$landesekunde;
}
```

Der Konstruktor ist also eine ”normale“ Methode, deren Namen dem Klassennamen entspricht. Er kann, wie jede Methode, Übergabeparameter akzeptieren. In unserer Raketenklasse ist die Initialisierung der Instanzvariablen Aufgabe des Konstruktors. Die Syntax der Implementierung des Konstruktors kennen wir aus dem Euro-Beispiel (Kapitel 8.2.1).

Dann folgt die Methode zur Berechnung der Flugzeit:

```
function berechneFlugzeit(&$flugzeitTage,
                          &$flugzeitStunden, &$flugzeitMinuten,
                          &$flugzeitSekunden)
{
```

```
        $starttag=$this->tageInMonaten($this->startmonat,$this->starttag);
        // nun landetage ausrechnen
        $landetag=$this->tageInMonaten($this->landemonat,$this->landetag);
        $startzeitInSekunden=$this->berechneSekunden
                ($starttag,$this->startstunde,$this->startminute,
                                                $this->startsekunde);
        $landezeitInSekunden=$this->berechneSekunden
                ($landetag,$this->landestunde,$this->landeminute,
                                                $this->landesekunde);
        // flugzeitInSekunden berechnen
        $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitSekunden<0)
        {
            return false;
        }
        $this->tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                            $flugzeitStunden, $flugzeitMinuten,
                            $flugzeitSekunden);
        return true;
    }
```

berechneFlugzeit ruft seinerseits weitere Methoden der Klasse auf. Der Methodenaufruf interner Methoden erfolgt analog zum Zugriff auf die Instanzvariablen. Dem Methodenname wird `$this->` vorangestellt.

Auffällig ist, dass die weiteren Methoden der Raketenklasse in Abb. 8.4 nicht dargestellt sind. Der Grund dafür ist, dass Entwickler, die die Raketenklasse nutzen wollen, diese Methoden nicht zu kennen brauchen. Denn wie die Berechnung der Flugzeit von unserer Klasse durchgeführt wird, ist für einen Benutzer der Klasse nicht von Bedeutung. Er benötigt nur das Ergebnis. Die in Abbildung 8.4 dargestellten Methoden sind die Schnittstelle der Klasse dargestellt.

In anderen objektorientierten Sprachen, wie z.B. Java kann man solche Methoden kenntlich machen und dadurch jede Benutzung außerhalb der Klasse selber verhindern. Die Implementierung der Methoden entspricht Kapitel 7.3.3.

Nun zur Benutzung der Klasse:

Beispiel 8.4 *Aufruf der Raketen-Klasse*

```
<!-- raketenbeispiel 5 des Textes
  Dateiname: raketen5.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
<?php
   //Funktionen einlesen
   require_once("./klassen/Rakete.inc.php");
?>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
```

```
    Die Flugzeit wird berechnet.<br> <hr>
<?php
  // Wir pruefen nun ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
        //erster Aufruf des Scripts wir muessen das Eingabeformular
        //praesentieren

        echo "<form name='raketen2' action='$PHP_SELF' method='post'>";
?>
        <table border>
            <tr>
                <td>
                    Startmonat
                </td>
                <td>
                    <input type="text" name="startmonat" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Starttag
                </td>
                <td>
                    <input type="text" name="starttag" size=12>
                </td>
            </tr>

            <tr>
                <td>
                    Startzeit Stunden
                </td>
                <td>
                    <input type="text" name="startStunden" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Startzeit Minuten
                </td>
                <td>
                    <input type="text" name="startMinuten" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Startzeit Sekunden
                </td>
                <td>
                    <input type="text" name="startSekunden" size=12>
                </td>
            </tr>
            <tr>
                <td>
```

```
                    Landemonat
                </td>
                <td>
                    <input type="text" name="landemonat" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landetag
                </td>
                <td>
                    <input type="text" name="landetag" size=12>
                </td>
            </tr>

            <tr>
                <td>
                    Landezeit Stunden
                </td>
                <td>
                    <input type="text" name="landeStunden" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landezeit Minuten
                </td>
                <td>
                    <input type="text" name="landeMinuten" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Landezeit Sekunden
                </td>
                <td>
                    <input type="text" name="landeSekunden" size=12>
                </td>
            </tr>
            <tr>
                <td colspan="2" align="center">
                    <input type="submit" name="Button1" value="Abschicken">
                </td>
            </tr>
        </table>
        </form>
<?php
    }
    else
    {
        $rakete=new Rakete($startmonat, $starttag, $startStunden,
                           $startMinuten, $startSekunden,
                           $landemonat, $landetag,
                           $landeStunden, $landeMinuten,
```

```
                        $landeSekunden);
        if(!($rakete->berechneFlugzeit($flugzeitTage,
                        $flugzeitStunden, $flugzeitMinuten,
                        $flugzeitSekunden)))
        {
            echo "Landezeit vor Startzeit";
            exit;
        }
        if($flugzeitTage==0)
        {
            echo ("Die Flugzeit betr&auml;gt: <br>" .
                        "$flugzeitStunden Stunden <br>" .
                        "$flugzeitMinuten Minuten <br>" .
                        "$flugzeitSekunden Sekunden <br>");
        }
        else
        {
            echo ("Die Flugzeit betr&auml;gt: <br>" .
                        "$flugzeitTage Tage <br>" .
                        "$flugzeitStunden Stunden <br>" .
                        "$flugzeitMinuten Minuten <br>" .
                        "$flugzeitSekunden Sekunden <br>");
        }
    }
?>
</body>
</html>
```

In den Zeilen

```
$rakete=new Rakete($startmonat, $starttag, $startStunden,
                    $startMinuten, $startSekunden,
                    $landemonat, $landetag,
                    $landeStunden, $landeMinuten,
                    $landeSekunden);
```

wird ein Raketenobjekt erzeugt. Danach wird unser Konstruktor aufgerufen. Der Konstruktor erhält als Übergabevariablen die von den Benutzern im Formular eingegebenen Werte. An der Erzeugung eines Objekts ändert sich auch dann, wenn ein Konstruktor implementiert ist, eigentlich nichts. Es ist lediglich darauf zu achten, dass die Übergabevariablen, die der Konstruktor erwartet, auch übergeben werden.

Danach rufen wir die Berechnungsmethode des Objekts auf:

```
if(!($rakete->berechneFlugzeit($flugzeitTage,
                    $flugzeitStunden, $flugzeitMinuten,
                    $flugzeitSekunden)))
{
    echo "Landezeit vor Startzeit";
    exit;
}
```

Die Methode berechneFlugzeit gibt `true` zurück, wenn die Berechnung erfolgreich war (Startzeit liegt vor Landezeit), im umgekehrten Fall wird `false` zurückgegeben. Daher überprüfen wir in einem `if`-Statement den Rückgabewert.

Die Ausgabe erfolgt dann wie in Kapitel 7.3.3.

An diesem Beispiel zeigt sich zum ersten Mal die Stärke der objektorientierten Entwicklung. Die Benutzer unserer Klasse müssen nur zwei Dinge wissen:

- Die Argumente des Konstruktors.
- Die Signatur von berechneFlugzeit.

8.3 Klassenmethoden

In der bisherigen Diskussion benötigten wir immer ein Objekt, um Methoden einer Klasse auszuführen (in obigen Beispielen mussten wir erst Objekte der Klassen Euros oder Rakete mit dem Schlüsselwort new erzeugen, erst daraufhin konnten wir die Methoden der Klassen nutzen). Es gibt aber Fälle, wo wir die Methoden einer Klasse nutzen wollen, aber ein Objekt der Klasse nicht unbedingt brauchen (dies ist z.B. immer dann der Fall, wenn es keine Instanzvariablen gibt).

Dies machen wir uns sofort an einem Beispiel klar. Betrachten wir Beispiel 8.3 erneut. Ganz perfekt ist dieses Beispiel immer noch nicht. Denn bei näherer Betrachtung enthält die Implementierung der Raketenklasse Datumfunktionen. Datumfunktionen kann man aber sicherlich auch noch in anderen Bereichen nutzen, nicht nur bei der Berechnung der Flugzeit einer Rakete. Einfache Beispiele sind Terminkalender oder Projektmanagementsysteme.

Wollten wir in einer anderen Anwendung aber die bereits programmierten Datumfunktionen nutzen, müssten wir jeweils eine Raketeninstanz erzeugen. Dies ist aber nicht wirklich schön, weil Raketen z.B. mit Terminkalendern nicht allzu viel gemeinsam haben. Wir haben also einen Fehler in der Modellierung, die Datumfunktionen haben in der Raketenklasse nichts verloren, sie gehören eigentlich in eine eigene Klasse, die dann von der Raketen-Klasse genutzt werden kann.

Eine Lösung ist also, die Datumfunktionen in eine eigene Klasse auszulagern und diese Klasse dann von der Raketen-Klasse aus zu nutzen. Die Implementierung der Datumsklasse ist dann folgendermaßen:

Beispiel 8.5 *Die Datumklasse*

```
<?php
class Datumfunktionen
// datumsfunktionen
// Datei:Datumfunktionen.inc.php
// Verzeichnis: klassen
{
    function tageInMonaten($monat, $tag)
    {
        switch($monat)
        {
            case 1:
                $tag=$tag; //ueberflussig, nur der Klarheit wegen
                break;
            case 2:
                $tag=31+$tag;
                break;
            case 3:
```

```
                $tag=31+28+$tag;
                break;
            case 4:
                $tag=31+28+31+$tag;
                break;
            case 5:
                $tag=31+28+31+30+$tag;
                break;
            case 6:
                $tag=31+28+31+30+31+$tag;
                break;
            case 7:
                $tag=31+28+31+30+31+30+$tag;
                break;
            case 8:
                $tag=31+28+31+30+31+30+31+$tag;
                break;
            case 9:
                $tag=31+28+31+30+31+30+31+31+$tag;
                break;
            case 10:
                $tag=31+28+31+30+31+30+31+31+30+$tag;
                break;
            case 11:
                $tag=31+28+31+30+31+30+31+31+30+31+$tag;
                break;
            case 12:
                $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
                break;
        }
        return $tag;
    }

    function berechneSekunden($tag, $stunden, $minuten, $sekunden)
    {
        return($tag*24*3600+
                    $stunden*3600+$minuten*60+$sekunden);
    }

    function tageStundenMinutenSekundenAusSekunden(&$tage,
                        &$stunden, &$minuten, &$sekunden)
    {
        // zuerst sekunden
        $minuten=floor($sekunden/60);
        $sekunden=$sekunden%60;
        //nun minuten und stunden
        $stunden=floor($minuten/60);
        $minuten=$minuten%60;
        //und tage
        $tage=floor($stunden/24);
        $stunden=$stunden%24;
    }
}
?>
```

Diese Klasse enthält keine Instanzvariablen und keinen Konstruktor. Sie entspricht im Wesentlichen Beispiel 7.21. Einziger Unterschied ist, dass die Klasse mit

```
class Datumfunktionen
{
```

eingeleitet und mit einer zusätzlichen schließenden Klammer beendet wird. Dies liegt aber daran, dass unsere Klasse Datumfunktionen wirklich nur Funktionen enthält und keine Instanzvariablen.

Für solche Fälle gibt es eine alternative Aufrufsyntax. Wir müssen kein Objekt der Klasse mit dem new-Operator bilden, sondern können die Methoden der Klasse direkt mit der Syntax NameDerKlasse::NameDerMethode aufrufen. Dies zeigt Beispiel 8.6.

Beispiel 8.6 *Aufruf der Datumklasse*

```
<?php
require_once
("/usr/local/httpd/htdocs/neuesScript/php/klassen/Datumfunktionen.inc.php");
class Rakete
{
// Raketenklasse
// Datei:Rakete2.inc.php
// Verzeichnis: klassen
    var $startmonat;
    var $starttag;
    var $startstunde;
    var $startminute;
    var $startsekunde;
    var $landemonat;
    var $landetag;
    var $landestunde;
    var $landeminute;
    var $landesekunde;

    /*
    *
    *    Konstruktor
    *
    */

    function Rakete($startmonat, $starttag, $startstunde,
                    $startminute, $startsekunde, $landemonat,
                    $landetag, $landestunde, $landeminute,
                    $landesekunde)
    {
        $this->startmonat=$startmonat;
        $this->starttag=$starttag;
        $this->startstunde=$startstunde;
        $this->startminute=$startminute;
        $this->startsekunde=$startsekunde;
        $this->landemonat=$landemonat;
        $this->landetag=$landetag;
```

```
        $this->landestunde=$landestunde;
        $this->landeminute=$landeminute;
        $this->landesekunde=$landesekunde;
    }

    function berechneFlugzeit(&$flugzeitTage,
                          &$flugzeitStunden, &$flugzeitMinuten,
                          &$flugzeitSekunden)
    {
        $starttag=Datumfunktionen::tageInMonaten($this->startmonat,$this->starttag);
        // nun landetage ausrechnen
        $landetag=Datumfunktionen::tageInMonaten($this->landemonat,$this->landetag);
        $startzeitInSekunden=Datumfunktionen::berechneSekunden
                ($starttag,$this->startstunde,$this->startminute,
                                                $this->startsekunde);
        $landezeitInSekunden=Datumfunktionen::berechneSekunden
                ($landetag,$this->landestunde,$this->landeminute,
                                                $this->landesekunde);
        // flugzeitInSekunden berechnen
        $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitSekunden<0)
        {
            return false;
        }
        Datumfunktionen::tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                          $flugzeitStunden, $flugzeitMinuten,
                          $flugzeitSekunden);
        return true;
     }
}
?>
```

In der Zeile

```
    $starttag=Datumfunktionen::tageInMonaten($this->startmonat,
                                                    $this->starttag);
```

sehen wir einen beispielhaften Aufruf von Methoden der Datumklasse.

Nun stellt sich natürlich die Frage: Warum machen wir das überhaupt? Man könnte ja auch einfach das mit der Klasse weglassen und wie in Kapitel 7 vorgehen, wo wir die Funktionen in eine eigene Datei ausgelagert haben und die Funktionen dann benutzt haben. War vom Aufruf her auch einfacher, weil wir NameDerKlasse:: dann weglassen können.

Es gibt allerdings gute Gründe, Funktionen in Klassen zu kapseln:

- Hält man sich an die Konvention, als Namen der Datei, in der eine Klasse abgespeichert ist, den Klassennamen zu wählen, ist aus dem Funktionsaufruf[10] ersichtlich, in welcher Datei eine Funktion abgespeichert ist.

- Der wichtigste Grund allerdings ist der Namensraum. Jeder Funktionsname darf in einer php-Anwendung nur einmal vorkommen. Bei großen Anwendungen, die in viele Dateien aufgeteilt

[10] Der Klassename und damit der Dateiname ist im Aufruf enthalten.

sind und auch noch von mehreren Entwicklern realisiert werden, ist dies nicht ganz einfach. Bei der Nutzung von Klassen fällt diese Einschränkung weg. Hier gilt: Jeder Funktionsname darf in einer php-Klasse nur einmal vorkommen. Selbiges gilt übrigens auch für Variablen.

In wirklich objektorientierten Programmiersprachen (wie z.B. Java) müssen Klassenmethoden besonders gekennzeichnet werden. Dies ist in php, wie Sie gesehen haben, nicht der Fall. php entscheidet anhand des Aufrufs, ob es sich um eine Klassen-, oder Objektmethode handelt. Folgende Realisierung funktioniert in php nämlich auch:

Beispiel 8.7 *Alternativer Aufruf der Datumklasse*

```
<?php
require_once("/usr/local/httpd/htdocs/neuesScript/php/klassen/Datumfunktionen.inc.php
class Rakete
{
// Raketenklasse
// Datei:Rakete.inc.php
// Verzeichnis: klassen
    var $startmonat;
    var $starttag;
    var $startstunde;
    var $startminute;
    var $startsekunde;
    var $landemonat;
    var $landetag;
    var $landestunde;
    var $landeminute;
    var $landesekunde;

    /*
    *
    *    Konstruktor
    *
    */

    function Rakete($startmonat, $starttag, $startstunde,
                    $startminute, $startsekunde, $landemonat,
                    $landetag, $landestunde, $landeminute,
                    $landesekunde)
    {
        $this->startmonat=$startmonat;
        $this->starttag=$starttag;
        $this->startstunde=$startstunde;
        $this->startminute=$startminute;
        $this->startsekunde=$startsekunde;
        $this->landemonat=$landemonat;
        $this->landetag=$landetag;
        $this->landestunde=$landestunde;
        $this->landeminute=$landeminute;
        $this->landesekunde=$landesekunde;
    }

    function berechneFlugzeit(&$flugzeitTage,
                              &$flugzeitStunden, &$flugzeitMinuten,
```

```
                            &$flugzeitSekunden)
    {
        $datum=new Datumfunktionen();
        $starttag=$datum->tageInMonaten($this->startmonat,$this->starttag);
        // nun landetage ausrechnen
        $landetag=$datum->tageInMonaten($this->landemonat,$this->landetag);
        $startzeitInSekunden=$datum->berechneSekunden
                ($starttag,$this->startstunde,$this->startminute,
                                                $this->startsekunde);
        $landezeitInSekunden=$datum->berechneSekunden
                ($landetag,$this->landestunde,$this->landeminute,
                                                $this->landesekunde);
        // flugzeitInSekunden berechnen
        $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
        if($flugzeitSekunden<0)
        {
            return false;
        }
        $datum->tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                            $flugzeitStunden, $flugzeitMinuten,
                            $flugzeitSekunden);
        return true;
     }
}
?>
```

Wir können also auch ein Datumobjekt erzeugen. Danach sind die Methoden der Klasse als Objektmethoden nutzbar. So etwas geht in anderen objektorientierten Programmiersprachen nicht.

Aufgabe 8.1 *Erstellen Sie eine objektorientierte Lösung zu Aufgabe 7.1.*

Aufgabe 8.2 *Erstellen Sie eine objektorientierte Lösung zu Aufgabe 7.2.*

8.4 Klassen und Objekte in JavaScript

Klassen und aus ihnen abgeleitete Objekte in JavaScript bestehen wie in jeder anderen objektorientierten Programmiersprache auch aus Daten und Funktionen. Die Daten können, wie auch in php, beliebigen Typs sein (auch wieder selbst Objekte). In JavaScript werden die Daten Eigenschaften (Properties) genannt, die Funktionen eines Objekts heißen wie in php Methoden.

Von JavaScript zur Verfügung gestellte Objekte

Das für den JavaScript-Entwickler mit Abstand wichtigste im Umgang mit JavaScript-Objekten ist die Nutzung der von JavaScript zur Verfügung gestellten Objekte. Hierbei handelt es sich im Einzelnen um:

- *Objekte, die das Entwickeln von JavaScript-Anwendungen erleichtern:* In JavaScript existiert beispielsweise ein Date-Objekt zum Arbeiten mit dem Datum oder ein String-Objekt zur Stringbearbeitung. Diese Objekte müssen, wie in php, aus Ihren Klassen erzeugt werden. Wie in php geschieht dies mit dem `new`-Kommando.

- *Interaktion mit dem Browser:*Jedes wichtige html-Element wird in JavaScript durch ein Objekt repräsentiert. So ist z.B. jedes html-Formular (<form>-Tag) mit einem JavaScript-Objekt verbunden. Über die Methoden und Eigenschaften der den html-Elementen zugeordneten Objekte lassen sich html-Seiten dynamisch verändern. Eines dieser Objekte haben wir bereits kennengelernt. Das `document`-Objekt repräsentiert den Inhalt eines Browser-Fensters (oder Frames). Mittels der Methode `write()` des `document`-Objekt können wir von JavaScript aus in das Browser-Fenster schreiben. Das `document`-Objekt verfügt darüberhinaus über Eigenschaften (Instanzvariablen) wie URL (die URL der dargestellten Seite) oder lastModified (das Datum der letzten Änderung des im Browser-Fenster dargestellten Dokuments), die bei der html-Programmierung sinnvoll einzusetzen sind. Solche Objekte müssen nicht erzeugt werden. Sie werden vom Browser erzeugt und stehen zur Verfügung[11].

Diese Objekte und ihr Einsatz in der praktischen JavaScript-Entwicklung werden in den folgenden Kapiteln eingehend besprochen.

Aufruf von Objekt-Methoden und Ansprechen von Objekt-Properties

Durch das bereits erlernte Arbeiten mit dem `document`-Objekt[12] wissen wir auch, wie Methoden der Objekte aufgerufen werden. Dies geschieht durch den Objekt-Namen, einen Punkt (.) und den Namen der Methode, wie z.B. in

```
document.write ("Der Betrag in Euro: " + euro + "<br>");
```

In JavaScript ist also der Punkt (.) das, was in php der Pfeil (->). ist[13]

Objekt-Eigenschaften werden auf identische Weise angesprochen. Auch hier wird der Name der Eigenschaft mittels eines Punktes (.) an den Objektnamen angeschlossen.

Erstellung eigener Objekte

Die Programmierung eigener Objekte hat in Client-Side JavaScript meines Erachtens nicht die große Bedeutung, da in html-Dateien eingebettete Scripts meistens doch nicht so große Anwendungen sind, dass sie einer wohlüberlegten Objektstruktur bedürfen. Anders kann dies sein, wenn JavaScript serverseitig zur Implementierung größerer Anwendungen genutzt wird.

Serverseitiges JavaScript wird jedoch von den wenigsten http-Servern unterstützt und ist hier auch nicht Thema. Daher werde ich auf diese Thematik nicht weiter eingehen.

Statische Methoden, Klassenmethoden

Wie in php, gibt es auch in JavaScript Fälle, wo wir die Methoden einer Klasse nutzen wollen, aber ein Objekt der Klasse nicht unbedingt brauchen (dies ist z.B. immer dann der Fall, wenn keine Eigenschaften benötigt werden). Diese Klassenmethoden heißen in JavaScript[14] statische Methoden (engl. static).

Klassenmethoden (oder statische Methoden[15]) werden in JavaScript vermittels der Notation:

[11] Sonst hätten wir das `document`-Objekt ja auch nicht so ohne weiteres benutzen können.

[12] Auch wenn Ihnen bis hierhin nicht klar war, dass Sie so schwierige Sachen gemacht haben, wie von JavaScript bereitgestellte Objekte zu nutzen.

[13] Der Punkt war in php nämlich bereits durch die Stringverkettung belegt, in JavaScript verkettet ja der +-Operator Strings.

[14] Wie übrigens auch in Java.

[15] Ich werde von nun an beide Bezeichnungen benutzen.

```
NameDerKlasse.NameDerMethode
```

aufgerufen. Auch dies ist anders als in php, wo ja der ::-Operator genutzt wird.

Ebenfalls im Unterschied zu php können wir Klassenmethoden nicht als Methoden von aus dieser Klasse erzeugter Objekte aufrufen. Dies liegt daran, dass Klassenmethoden in JavaScript ausdrücklich als solche gekennzeichnet sind. Ob eine Methode eine Klassenmethode ist, muss der JavaScript-Dokumentation entnommen werden. Klassenmethoden sind dort als `static` gekennzeichnet (vgl. Abb. 8.5).

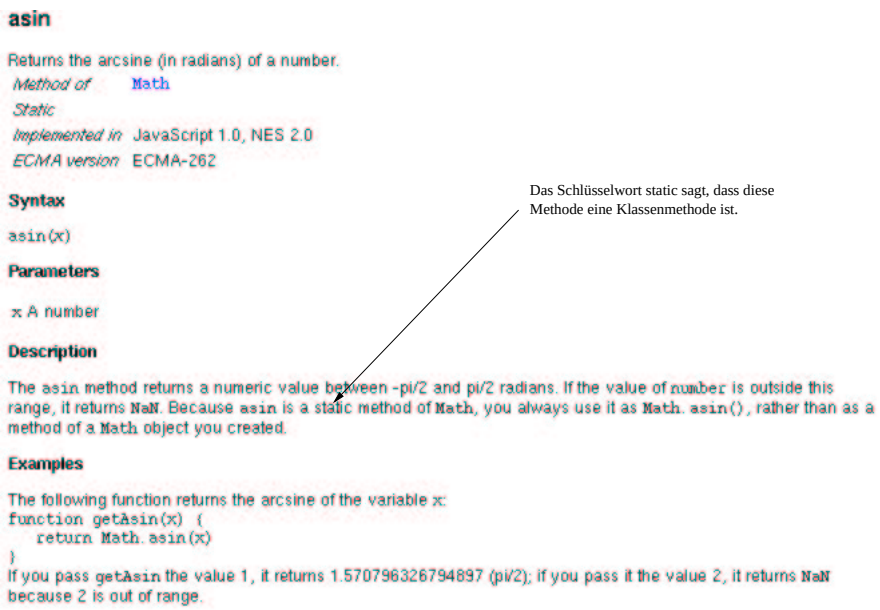


Abbildung 8.5: Erkennung statischer Methoden

Betrachten wir nun als Beispiel die JavaScript interne Klasse Math. In Math sind die mathematischen Funktionen, die JavaScript unterstützt, zusammengefasst. Nehmen wir nun an, wir wollten die Wurzel einer Zahl berechnen. Beispiel 8.8 zeigt eine mögliche Lösung:

Beispiel 8.8 *Nutzung statischer Methoden in JavaScript*

```
<!-- Programm zur Demonstration statischer Methoden
  Dateiname: matheRichtig.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Statische Methoden</title>
</head>
<body>
    <script language = "JavaScript">
        var eingabe;
        var ergebnis;

        eingabe = parseFloat(prompt("Bitte geben Sie die Zahl ein, "+
                    "deren Wurzel Sie wissen wollen!", ""));
        ergebnis = Math.sqrt(eingabe);
```

```
        document.write("Die Wurzel ist: " + ergebnis + "<BR>");
    </script>
</body>
</html>
```

Die Zeile

```
ergebnis=Math.sqrt(eingabe);
```

zeigt den Aufruf der statischen Methode. Wie oben beschrieben erfolgt dies, indem wir zunächst den Namen der Klasse (Math), dann den Punkt (.) und dann den Namen der statischen Methode (sqrt) schreiben. Beispiel 8.8 zeigt darüberhinaus, dass Funktions- und damit auch Methodenaufrufe schachtelbar sind:

```
eingabe=parseFloat(prompt("Bitte geben Sie die Zahl ein, "+
            "deren Wurzel Sie wissen wollen!", ""));
```

Die Abarbeitung erfolgt von innen nach außen.

Abb. 8.6 zeigt einen beispielhaften Programmlauf.

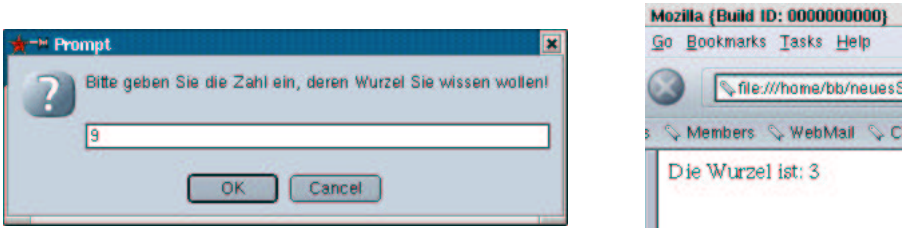


Abbildung 8.6: Ein- und Ausgaben von Beispiel 8.8

Wollten wir hingegen ein Objekt vom Typ Math erzeugen und dann sqrt als Objektmethode nutzen, würde sich unser Programm in folgender Weise ändern:

Beispiel 8.9 *Statischer Methoden als Objektmethoden in JavaScript*

```
<!-- Programm zur Demonstration statischer Methoden 2
  Dateiname: matheFalsch.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Statische Methoden</title>
</head>
<body>
    <script language = "JavaScript">
        var matheObjekt;
        matheObjekt = new Math();
        var eingabe;
        var ergebnis;
        eingabe = parseFloat(prompt("Bitte geben Sie die Zahl ein, "+
                    "deren Wurzel Sie wissen wollen!"));
        ergebnis = matheObjekt.sqrt(eingabe);
        document.write("Die Wurzel ist: " + ergebnis + "<BR>");
```

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

In php würde das funktionieren, in JavaScript geht das nicht. Beispiel 8.9 führt zu der in Abb. 8.7 dargestellten Fehlermeldung.

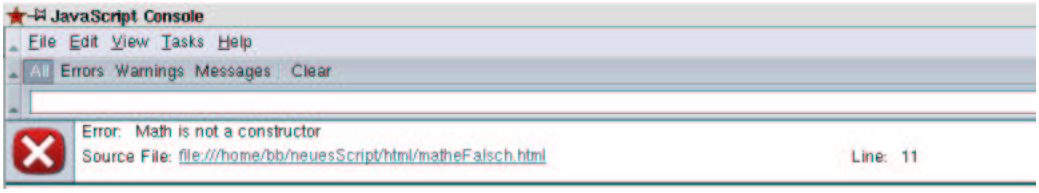


Abbildung 8.7: Fehlermeldung von Beispiel 8.9

Kapitel 9

JavaScript-Objekte und das Event-Modell

9.1 Das Window-Objekt und die Objekthierarchie

Das Browser-Fenster, in dem die html-Seite, die den JavaScript-Code enthält, dargestellt wird, wird durch ein window-Objekt repräsentiert.

Das window-Objekt verfügt über Methoden wie prompt() (haben wir ja bereits kennengelernt) oder alert() und confirm() (werden im folgenden besprochen).

Darüber hinaus hat das window-Objekt Eigenschaften, die sich von JavaScript aus steuern lassen. Dazu gehört die Statuszeile (status-Eigenschaft), die URL, die gerade dargestellt wird (location) oder die URL's der Seiten, die bis jetzt dargestellt wurden (history).

In Wirklichkeit sind sogar alle anderen JavaScript-html-Objekte entweder selbst Eigenschaften des window-Objekts oder Eigenschaften von Eigenschaften des window-Objekts (auch das document-Objekt, welches wir bereits öfter zum Schreiben in das Browser-Fenster benutzt haben).

Das window-Objekt ist die Wurzel (root) der JavaScript-Objekt-Hierarchie. Abb. 9.1 zeigt einen Auszug aus der JavaScript-Objekt-Hierarchie. Die dort dargestellten Objekte werden in den folgenden Kapiteln besprochen.

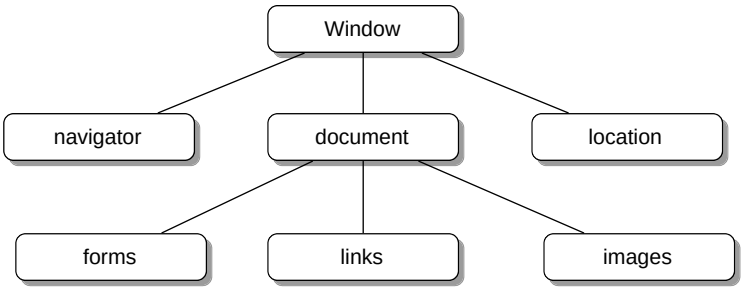


Abbildung 9.1: Die JavaScript-Klassen-Hierarchie

Abb. 9.1 ist folgendermaßen zu lesen: Das form-Objekt ist beinhaltet im document-Objekt, dies wiederum ist Teil des window-Objekts.

Benutzer-Interaktion mittels des window-Objekts

Das window-Objekt verfügt über drei Methoden, mittels derer es mit dem Benutzer interagieren kann. Die erste (prompt()) haben wir bereits kennengelernt. prompt() öffnet ein eigenes Fenster oberhalb des Browser-Fensters. Das Fenster enthält 2 Bereiche: Im ersten Bereich wird eine Meldung dargestellt (erster Parameter von prompt()), im zweiten Bereich kann der Benutzer Eingaben tätigen (der zweite Parameter von prompt() ist eine Vorgabe im Eingabefeld).

confirm() erwartet vom Benutzer eine Bestätigung. Der einzige Parameter von confirm() ist eine Meldung, die im von confirm() aufgespannten Fenster dargestellt wird. Der Benutzer kann mit einem OK-Button bestätigen (confirm() liefert dann true zurück) oder mit einem "Cancel-Button" abbrechen (confirm() liefert false zurück).

alert() gibt ein Fenster mit einem Hinweis aus. Das Fenster kann mittels eines OK-Buttons geschlossen werden.

Die von prompt() und confirm() geöffneten Fenster sind modal. Dies bedeutet, der laufende JavaScript-Code wird angehalten, bis der Benutzer seine Eingabe getätigt hat. Das Programm ist geblockt. alert() hingegen blockiert das Programm nicht.

Beispiel 9.1 zeigt eine ziemlich unsinnige Nutzung dieser Methoden.

Beispiel 9.1 *Window-Objekt, Interaktion*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Demonstration der window-Interaktion
  Dateiname: windowInteraktion.html //-->
<html>
<head>
  <title>window-Interaktion</title>
</head>
<body>
  <script language="JavaScript">
    var confirmMessage="Wollen Sie wirklich?";
    var alertMessage="Nun ist's passiert!";
    var promptMessage="Eingeben, egal was!";
    var eingabe;
    var entscheidung;
    eingabe=prompt(promptMessage,"");
    entscheidung=confirm(confirmMessage);
    if (entscheidung)
    {
        document.write("Sie haben OK geclickt!");
    }
    else
    {
        document.write("Sie haben Cancel geclickt!");
    }
    alert(alertMessage)
  </script>
</body>
</html>
```

Dies führt zu der in Abb. 9.2 gezeigten Darstellung:

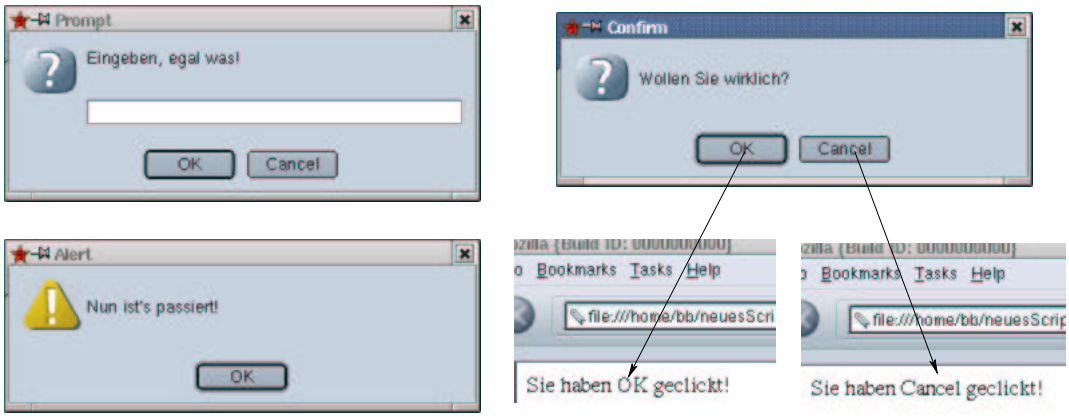


Abbildung 9.2: Interaktionsmethoden des window-Objekts

Die Nutzung der prompt-Methode erläutere ich nicht mehr, wir haben sie ja nun oft genug besprochen. confirm entspricht im Wesentlichen prompt, der einzige Unterschied ist, dass wir den Wert den confirm zurückgibt, nicht frei wählen können[1]. Wenn die OK-Schaltfläche betätigt wird, gibt confirm true zurück, bei Betätigung der Cancel-Schaltfläche[2] false. confirm ist also die richtige Wahl für Nachfragen im Stil von: „Wollen Sie das wirklich tun?“. In Abhängigkeit vom Verhalten des Nutzers kann das Programm dann reagieren. Standardbeispiel hierfür ist das Zurücksetzen von Formularen. Wie Sie ja wissen, gibt es dort die Möglichkeit, reset-Buttons zu erzeugen, die, wenn Sie geclickt werden, alle input-Felder auf den Standardwert (im Normalfall ungefüllt) zurücksetzen. Gerade bei großen Formularen ist es ganz nett, das Programm noch mal nachfragen zu lassen, ob das Löschen aller Formularfelder wirklich beabsichtigt ist und, wenn der Benutzer erschreckt meint „lieber nicht“, die Felder auch gefüllt zu lassen. confirm() wird daher meist im Zusammenhang mit `if` benutzt.

In Beispiel 9.1 wird als Text ausgegeben, welchen Button der Benutzer betätigt hat.

alert blendet, wie man sieht, einfach eine Nachricht auf, die man dann “wegclicken” muss.

Erstaunlich an Beispiel 9.1 (und eigentlich auch an den vorhergegangenen Nutzungen von prompt(), confirm(), alert () und document.write() ist, dass wir den Namen des window-Objektes nicht voranstellen müssen. Sie haben ja in Kapitel 8 gelernt, dass der Aufruf einer Methode eines Objekts in JavaScript mittels Objektname, Punkt, Methodenname erfolgt. Dies ist hier ja nicht der Fall. Der Grund, dass der Code aus Beispiel 9.1 dennoch funktioniert ist einfach, dass JavaScript automatisch, sofern kein Name eines window-Objekts angegeben ist, den Namen des aktuellen Fensters voranstellt. Der Name des aktuellen Fensters ist einfach window. Wir können dies in unseren Programmen aber auch selber tun. Beispiel 9.1 ist äquivalent zu 9.2:

Beispiel 9.2 *Window-Objekt, Interaktion mit Referenz des window*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Demonstration der window-Interaktion
  Dateiname: windowInteraktion2.html //-->
<html>
<head>
  <title>window-Interaktion</title>
</head>
```

[1] sieht man auch am fehlenden Eingabefeld.

[2] Ein echter Transfer ist, wenn Sie sich überlegen, was wohl bei Browsern mit deutscher Spracheinstellung passiert :-).

```
<body>
  <script language="JavaScript">
    var confirmMessage="Wollen Sie wirklich?";
    var alertMessage="Nun ist's passiert!";
    var promptMessage="Eingeben, egal was!";
    var eingabe;
    var entscheidung;
    eingabe=window.prompt(promptMessage,"");
    entscheidung=window.confirm(confirmMessage);
    if (entscheidung)
    {
        window.document.write("Sie haben OK geclickt!");
    }
    else
    {
        window.document.write("Sie haben Cancel geclickt!");
    }
    window.alert(alertMessage)
  </script>
</body>
</html>
```

In Beispiel 9.2 haben wir allen Methodenaufrufen eine explizite Referenz des aktuellen window-Objekts hinzugefügt. JavaScript erkennt, dass in allen Methoden-Aufrufen Referenzen auf ein window-Objekt existieren und fügt daher selbst keine mehr hinzu.

Übrigens sind alle bisher kennengelernten Funktionen, parseInt() oder parseFloat() Methoden des window-Objekts. An Beispiel 9.2 erkennen wir auch, wie wir Methoden von Objekt-Eigenschaften (gemeint ist: Objekte, die Teil anderer Objekte sind, vgl. Abb. 9.1) eines Objekts ansprechen können. Dies geschieht einfach mittels der Syntax:

```
Objektname.Objektname.Methodenname()
```

In Beispiel 9.2 sehen wir dies anhand der Nutzung der write-Methode des document-Objekts des Windows-Objekts

```
window.document.write("Sie haben OK geclickt!");
```

Das window-Objekt enthält Darüber hinaus Methoden, um weitere Browserfenster zu öffnen oder bestehende Browserfenster zu schließen[3]. JavaScript erlaubt uns z.B. auch, in neu geöffnete weitere Browserfenster zu schreiben. Wir müssen nur den Namen des Fensters wissen (eine Referenz zum Fenster haben) und können dann vermittels

```
Fenstername.document.write()
```

in dieses Fenster schreiben. Beispiel 9.3 zeigt dies!

Beispiel 9.3 *Window-Objekt, Öffnen eines neuen Fensters*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Oeffnen eines neuen Fensters
```

[3] Auch das eigene Fenster kann man mit JavaScript schließen, auch wenn das nicht ganz so sinnvoll ist.

```
  Dateiname: windowOpen.html //-->
<html>
<head>
  <title>Neues Fenster&ouml;ffner</title>
</head>
<body>
    <p>
        Ein neues Fenster wird ge&ouml;ffnet!
    </p>
    <script language="JavaScript">
        var neuesFenster=window.open("", "neu");
        neuesFenster.document.write("<html>");
        neuesFenster.document.write("<head>");
        neuesFenster.document.write("<title>Neues Fenster</title>");
        neuesFenster.document.write("</head>");
        neuesFenster.document.write("<body>");
        neuesFenster.document.write("<h2> Zu schreibender  Text! </h2>");
        neuesFenster.document.write("<p> Zweite Zeile </p>");
        neuesFenster.document.write("</body>");
        neuesFenster.document.write("</html>");
    </script>
</body>
</html>
```

Dies führt zu der in Abb. 9.3 gezeigten Darstellung:

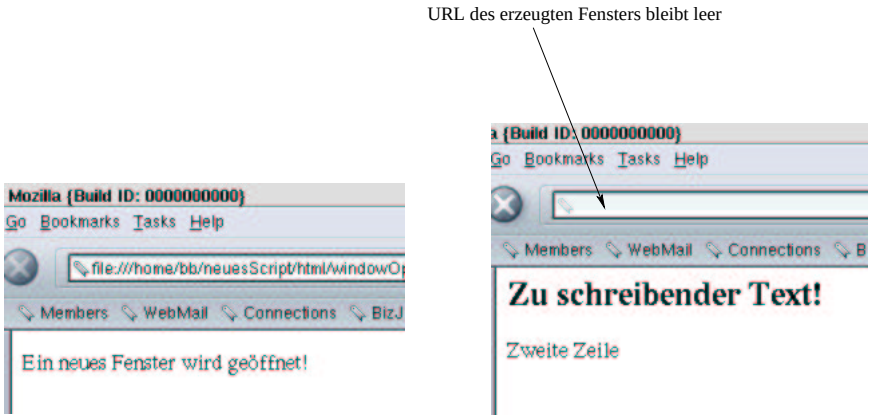


Abbildung 9.3: Öffnen eines neuen Fensters

In der Zeile

```
var neuesFenster=window.open("", "neu");
```

wird eine neues Fenster geöffnet. Eine Referenz wird der Variable neuesFenster zugewiesen. Variablen können also nicht nur Strings und Zahlen, sondern auch kompliziertere Dinge, wie Objekte, enthalten.

Die Methode open() des Window-Objekts öffnet also ein neues Fenster. Diese Methode erwartet mindestens zwei Übergabeparameter. Der erste ist unserem Fall leer (repräsentiert durch “”). Als erster Parameter könnte eine URL angegeben werden. Der Browser würde dann diese URL in das neue

Fenster laden. Der zweite Parameter ist der Name des Fensters. Wir könnten noch weitere Übergabeparameter angeben, die sich dann auf das Aussehen des Fensters auswirken würden[4].

Über die Variable neuesFenster können wir das Fenster nun ansprechen:

```
neuesFenster.document.write("<html>");
neuesFenster.document.write("<head>");
```

Selbstverständlich muss html in das neue Fenster geschrieben werden. Das URL-Feld des neuen Fensters bleibt, wie auch in Abb. 9.3 sichtbar, leer[5].

Dieses „Feature“ sollte man meiner Ansicht nach sehr vorsichtig behandeln. Jedes neue Fenster nimmt Platz auf dem Workspace der Benutzer weg. Dies macht den Bildschirminhalt für die Benutzer schnell sehr unübersichtlich. In Kapitel ?? werde ich im Zusammenhang mit dem navigator-Objekt eine sinnvolle Anwendung vorstellen.

Neben den bisher behandelten gibt es weitere Eigenschaften und Methoden des window-Objekts. Teilweise werden sie in folgenden Kapiteln im Zusammenhang mit den anderen, den Browser steuernden, JavaScript-Objekten besprochen. Für die restlichen verweise ich auf die Dokumentation.

9.2 Das document-Objekt

Das document-Objekt repräsentiert den Schreibbereich des Browser-Fensters. Seine wichtigste Methode ist die write()-Methode. Wir haben die write()-Methode schon sehr häufig angewendet, so dass sich eine weitere Diskussion hier erübrigt.

Neben der write()-Methode existiert eine writeln()-Methode. writeln() fügt einen Zeilenvorschub an. Dies ist, wenn wir html schreiben, nur bedingt sinnvoll, da Zeilenvorschübe von Browsern ja ignoriert werden. Man kann in JavaScript allerdings auch nicht-html-Dokumente (z.B. Ascii-Dokumente) öffnen. Dann wird die writeln()-Methode wieder sinnvoll.

Daneben enthält das document-Objekt einige ganz nützliche Eigenschaften, z.B:

- *lastModified:* Enthält das Datum der letzten Änderung.
- *URL:* Enthält den URL des Dokuments.
- *title:* Enthält den Titel des Dokuments (Text zwischen den ¡title¿ und ¡/title¿ Tags.
- *referrer:* Der URL der Seite von der der Benutzer kam.
- *bgColor, fgColor, linkColor, alinkColor:* Für das Dokument eingestellte Farben.

Beispiel 9.4 zeigt den sinnvollen Einsatz der Eigenschaft lastModified. Wir wollen das Datum der lezten Änderung in alle Dateien einer Web-Site aufnehmen. Darüber hinaus wird hier die Benutzung von nicht vom Browser erstellten Objekten am Beispiel des date-Objekts demonstriert. Die Funktion zum Ermitteln des Datums der letzten Änderung wird in eine eigene Datei ausgelagert. So können wir diese Funktion für alle Seiten unseres Internet-Auftritts nutzen. Zunächst also die html-Datei:

Beispiel 9.4 *document-Objekt, Datum der letzten Änderung*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Darstellung des Datums
    der letzten Aenderung
```

[4] Z.B. Höhe, Breite, welche Bedienleisten eingeblendet werden

[5] Ist ja auch klar, das neue Fenster stellt keine URL da.

```
     Dateiname: letzteAenderung.html //-->
<html>
<head>
  <title>Datum letzte &Auml;nderung</title>
</head>
<body>
  <p>
    Dieses Dokument enth&auml;lt irgendeinen Text!
  </p>
  <font size=2>
  <p align="right">
    <script language="JavaScript"
            src="./javascript/datumLetzteAenderung.js">
    </script>
</body>
</html>
```

Nun der JavaScript-Code in seiner Datei:

Beispiel 9.5 *document-Objekt, Datum der letzten Änderung JavaScript-Code*

```
// Dateiname: datumLetzteAenderung.js

var dateChanged=new Date(document.lastModified);
var day = dateChanged.getDate();
var month = germanMonth(dateChanged.getMonth());
var year = dateChanged.getFullYear();
document.write("Datum der letzten &Auml;nderung: ");
document.write(day + " " + month + ". " + year);

function germanMonth(monthAsInteger)
{
    if (monthAsInteger == 0)
    {
        return "Jan";
    }
    if (monthAsInteger == 1)
    {
        return "Feb";
    }
    if (monthAsInteger == 2)
    {
        return "M&auml;r";
    }
    if (monthAsInteger == 3)
    {
        return "Apr";
    }
    if (monthAsInteger == 4)
    {
        return "Mai";
    }
    if (monthAsInteger == 5)
    {
```

```
        return "Jun";
    }
    if (monthAsInteger == 6)
    {
        return "Jul";
    }
    if (monthAsInteger == 7)
    {
        return "Aug";
    }
    if (monthAsInteger == 8)
    {
        return "Sep";
    }
    if (monthAsInteger == 9)
    {
        return "Okt";
    }
    if (monthAsInteger == 10)
    {
        return "Nov";
    }
    if (monthAsInteger == 11)
    {
        return "Dez";
    }
}
```

Das Datum der letzten Änderung ist im Attribut lastModified des document-Objekts abgespeichert. Also wird es mit

```
document.lastModified
```

angesprochen. Das Datum wird allerdings im amerikanischen Format (dd/mm/yyyy) vorgehalten. Um eine „schöne“ Ausgabe zu erhalten, müssen wir es also umformatieren. Hierfür biete sich die Nutzung des date-Objekts an.

Zunächst wird aus dem String, den document.lastModified zurückgibt, ein Date-Objekt erzeugt. Wie bereits dargestellt werden neue Objekte mit dem Schlüsselwort new() aus ihren Klassen erzeugt. Dem Konstruktor wird das Datum, zu dem das Objekt erzeugt werden soll, als String übergeben. Nun können die Methoden der Date-Klasse eingesetzt werden. getDate() gibt den Tag der letzten Änderung zurück, getMonth() den Monat und getFullYear() das Jahr (4-stellig). Wie immer werden die Methoden der Objekte vermittels eines Punktes an den Objektnamen angeschlossen.

Zur “schönen” Darstellung des Monats dient die Funktion germanMonth(). getMonth() liefert den Monat als Zahl zwischen 0 und 11 zurück, germanMonth() wandelt dies durch eine Abfolge von `if`-Anweisungen in den Namen des Monats um (vgl. Abb. 9.4).

9.3 Event-Handler

Bislang wurde JavaScript-Code in html-Seiten genau dann ausgeführt, wenn der Browser im `<body>`-Teil auf den `<script language=Javascript>`-Tag gestoßen ist. Der JavaScript-Code startet

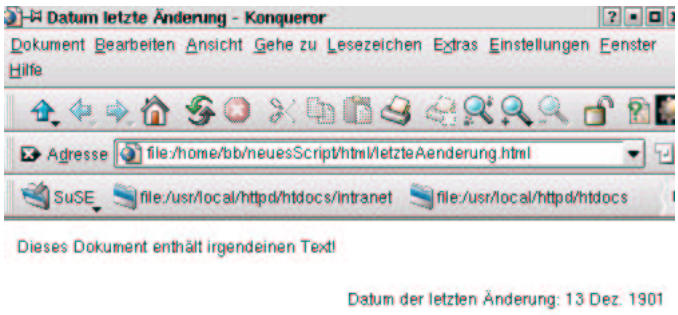


Abbildung 9.4: Letzte Änderung eines html-Dokuments

also ohne Interaktion mit dem Benutzer. Der Benutzer kann auch nicht entscheiden, welcher Teil des Codes ausgeführt wird.

Verglichen mit heutigen Oberflächen, ist dies “Technik von gestern”. Vergegenwärtigen Sie sich Ihre Umgehensweise mit Rechnern. Wenn ein Computer mit einer grafischen Oberfläche startet, wartet das Betriebssystem nach Ihrer Anmeldung auf eine Aktion Ihrerseits. Sie starten z.B. ein Textverarbeitungsprogramm. Dies tun Sie z.B. dadurch, dass Sie auf das Icon des Programms klicken. Nach dem Start der Textverarbeitung wartet das Programm auf weitere Eingaben von Ihnen. Sie klicken z.B. auf das Öffnen-Icon, um eine Datei zu laden. Daraufhin führt die Textverarbeitung den zugehörigen Code aus und präsentiert Ihnen eine Datei-Öffnungsbox.

Bei jeder Aktion Ihrerseits wird ein Event (Ereignis) ausgelöst. Dann wird der zu diesem Event gehörige Code durchgeführt. Dieselbe Möglichkeit bietet auch die clientseitige html-JavaScript-Kombination.

html stellt grafische Interaktionselemente bereit. Dies sind z.B. Links, auf die die Benutzer clicken können und damit den Browser veranlassen, eine neue Seite zu laden. Dies sind aber auch Eingabefelder oder Button in Formularen. Abb. 9.5 zeigt eine Auswahl grafischer html-Interaktionselemente. Die Benutzer können hier in den Textfeldern Eingaben machen und Button klicken, um Berechnungen zu starten.

Eine Lösung zu Abb. 9.5 können wir derzeit nur in php realisieren. php kann aber nur auf einen Event reagieren,[6] nämlich dann, wenn die Seite übertragen wird. Dies bringt aber Nachteile mit sich:

- Wenn der Benutzer eine Fehleingabe tätigt, z.B. in Beispiel 9.5 als Operanden keine Zahlen eingibt, fällt dies erst nach der Übertragung an den Server auf. Das php-Script auf dem Server muss also eine Fehlerseite erzeugen und an den Browser des Benutzers zurückübertragen. Die korrekten Eingaben sollten wieder dort erscheinen, wo der Benutzer sie eingetragen hat. Zusätzlich muss eine Meldung erzeugt werden, die den Fehler beschreibt und sagt, wie man es im zweiten Anlauf besser machen könnte. So etwas ist nicht ganz so einfach zu realisieren, belastet das Netz und erzeugt für den Benutzer unnötige Wartezeiten. Besser ist, die Überprüfungen in JavaScript auf dem Client zu machen und die Inhalte der Felder erst dann an den Server zu übertragen, wenn sie plausibel sind.

- Eingabekontrollen können erst ganz zum Schluss durchgeführt werden, wenn der Benutzer alle seine Eingaben getätigt hat. Besser wäre, nach jeder Eingabe gleich auf Korrektheit zu überprüfen.

[6] Was so strenggenommen auch nicht stimmt, php ist eine serverseitige Anwendung und reagiert clientseitig auf gar keinen Event. Durch clicken eines Submit-Button wird der Browser angewiesen, die Inhalte der html-Seite zum Server zu übertragen. Wir können es dann so einrichten, dass ein php-Programm die Eingaben der Benutzer erhält und weiterverarbeitet.

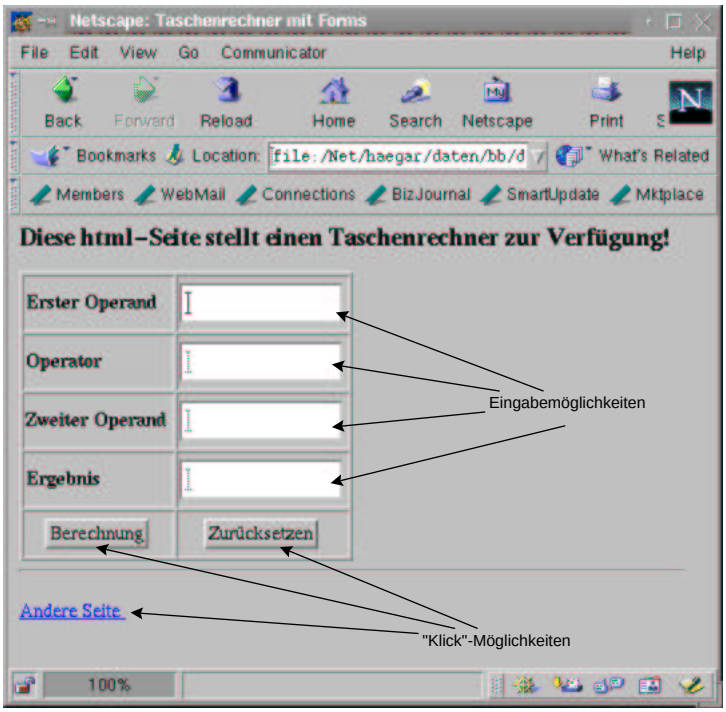


Abbildung 9.5: Events in JavaScript

Allerdings können wir mit unserem bisherigen Wissen nicht feststellen, ob und wann der Benutzer die grafischen Interaktionselemente bedient. Um dieser Problematik abzuhelfen, besitzen alle grafischen html-Interaktionselemente Event-Handler, aus denen heraus JavaScript-Code ausgeführt werden kann. Dies werden wir in den folgenden Kapiteln besprechen. Event-Handler besitzen eine weitere Besonderheit: Aus Event-Handlern darf document.write() nicht benutzt werden. Dies leuchtet aber auch ein, da der Browser die Seite bereits vollständig dargestellt hat[7], wenn Benutzer Events auslösen können. Es wäre also völlig unklar, wohin document.write() schreiben sollte.

9.4 Das link-Objekt

Beginnen wollen wir mit Event-Handlern des Link-Objekts. Dies ist zwar nicht das wichtigste aller Beispiele, Formularüberprüfungen spielen m.E. eine wichtigere Rolle, jedoch kann man sich das Verhalten von Event-Handlern am Link-Objekt gut veranschaulichen.

Jeder Link in einer html-Datei wird von JavaScript als Objekt behandelt und kann von JavaScript angesprochen werden. Für das Link-Objekt sind u.a. folgende Event-Handler definiert:

- onMouseOver.
- onMouseOut.
- onClick.

onMouseOver wird ausgelöst, wenn der Benutzer die Maus auf den Link bewegt, onMouseOut, wenn die Maus vom Link wegbewegt wird, onClick, wenn der Benutzer auf den Link klickt. Beginnen wir mit einem Beispiel zur Nutzung des onMouseOver Event-Handlers.

[7] Unter anderem hat er das End-Tag `</html>` bereits berabeitet

Beispiel 9.6 *Demonstration des onMouseOver Event-Handlers*

```
<!-- Programm zur Demonstration des onMouseOver
    Event-Handlers
    Dateiname: addition1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>On Mouse Over</title>
</head>
<body>
    <a href="www.fh-bochum.de" onMouseOver=
                "prompt('Geben Sie etwas ein!','')">
                Zur Fachhochschule Bochum
    </a>
</body>
</html>
```

Ein Event-Handler wird also in das entprechende html-Tag aufgenommen:

```
<a href="www.fh-bochum.de" onMouseOver=
            "prompt('Geben Sie etwas ein!','')">
```

Die Aktionen, die ausgeführt werden sollen, wenn das Ereignis (Event) des Event-Handlers eintritt, werden an diesen nach einem Gleichheitszeichen angeschlossen. Sie werden in Anführungszeichen eingeschlossen. Beachten Sie, dass wir hier, da die gesamte prompt-Methode in Anführungszeichen steht, innerhalb von prompt Hochkommata benutzen müssen. In Beispiel 9.6 wird also jedesmal, wenn der Benutzer die Maus auf den Link führt, ein Eingabefenster geöffnet, in dem “Geben Sie irgendwas ein!” steht. Abb. 9.6 zeigt dies.

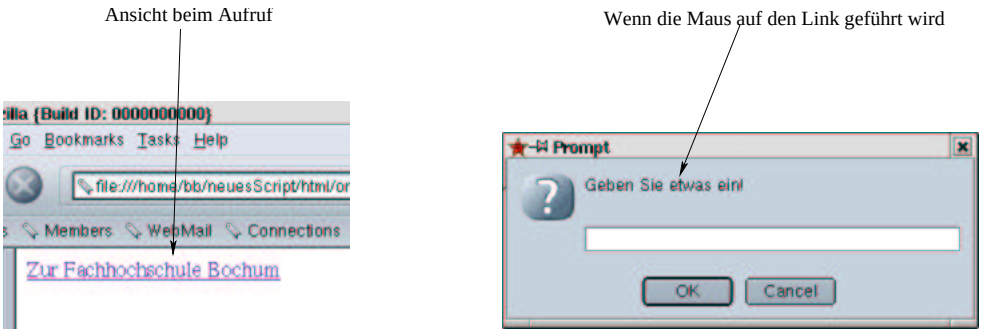


Abbildung 9.6: Der onMouseOver-Event des Link-Objekts

Die Aktionen eines Event-Handlers können beliebige JavaScript-Kommandos, -Methoden oder Funktionen (natürlich auch selbstgeschriebene) sein.

Als nächstes Beispiel betrachten wir den onClick Event-Handler:

Beispiel 9.7 *Demonstration des onClick Event-Handlers des Link-Objekts*

```
<!-- Programm zur Demonstration des onClick
```

```
    Event-Handlers
    Dateiname: linkOnClick.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>On Click Event</title>
  <script language="JavaScript">
        function frage()
        {
            if (confirm ("Sie verlassen diesen Server!"))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
  </script>
</head>
<body>
<h2>
Dies Programm fragt beim Verlassen des Servers nach!
</h2>
<p>
<a href= "http://www.mfh-iserlohn.de" onClick =
                    "return(frage())";>
MFH
</a>
</p>
</body>
</html>
```

Beispiel 9.7 nutzt eine Eigenschaft des onClick-Event-Handlers aus. Gibt der onClick-Event-Handler true zurück, wird der Link ausgeführt (die im Link genannte Seite wird geladen), gibt er false zurück, wird der Link nicht ausgeführt[8]. In Beispiel 9.7 wird auf die Leitseite der MFH verwiesen. Klickt der Benutzer auf den Link, wird die Funktion frage() ausgeführt. frage() bringt ein confirm-Fenster auf den Bildschirm, welches nachfragt, ob der Server wirklich verlassen werden soll (wir nehmen an, wir befinden uns auf einer Seite der Fachhochschule Bochum). Klickt der Benutzer auf den OK-Button des confirm-Fensters, gibt frage() true zurück und der Link wird ausgeführt, klickt er Cancel, gibt frage() false zurück und die Seite wird nicht verlassen.

Beachten Sie das `return`-Kommando im Event-Handler. Um oben beschriebenes Verhalten zu erzeugen, muss dieses `return` dort stehen! Beispiel 9.7 lässt sich allerdings auch verkürzt programmieren:

Beispiel 9.8 *Demonstration des onClick Event-Handlers des Link-Objekts: Verkürzte Fassung*

```
<!-- Programm zur Demonstration des onClick
    Event-Handlers
```

[8] Dies ist übrigens bei fast allen Objekten, die über einen onClick-Event-Handler verfügen, so.

```
    Dateiname: linkOnClick2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>On Click Event</title>
</head>
<body>
<h2>
Dies Programm fragt beim Verlassen des Servers nach!
</h2>
<p>
<a href= "http://www.mfh-iserlohn.de" onClick =
                    "return(confirm('Sie verlassen diesen Server!'))";>
MFH
</a>
</p>
</body>
</html>
```

Abb. 9.7 zeigt das Verhalten von Beispiel 9.7 und 9.8.

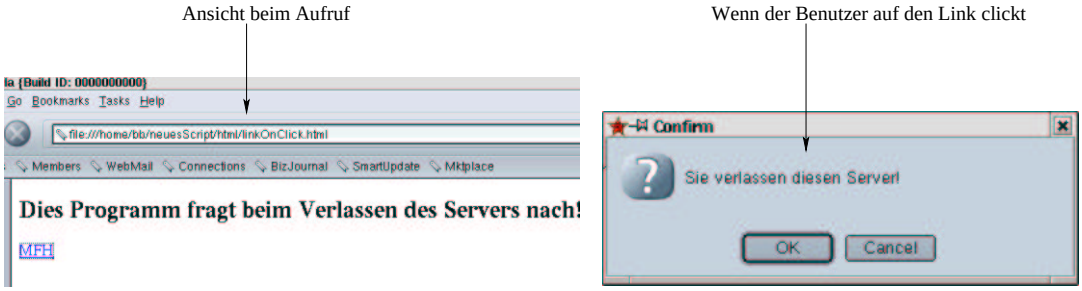


Abbildung 9.7: Der onClick-Event des Link-Objekts

Im nächsten Beispiel setzten wir drei Event-Handler des Link-Objekts ein:

Beispiel 9.9 *Demonstration von Event-Handlern des Link-Objekts:*

```
<!-- Programm zur Demonstration von
    Event-Handlern des Link-Objekts
    Dateiname: linkEvents.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Event-Handler Link-Objekt</title>
  <script language="JavaScript">
    function frage()
    {
        var endung;
        var meinLink = "http://www.firma";
        endung = prompt("Geben Sie Ihre Top Level Domain an!", "de");
        switch (endung)
        {
```

```
            case "com":
                meinLink = meinLink + ".com";
                break;
            case "edu":
                meinLink = meinLink + ".edu";
                break;
            case "gov":
                meinLink = meinLink + ".com";
                break;
            case "uk":
                meinLink = meinLink + ".uk";
                break;
            default:
                meinLink = meinLink + ".de";
        }
        return meinLink;
    }
  </script>
</head>
<body>
<h2>
Dies Programm verzweigt zum richtigen FTP-Server!
</h2>
<p>
        <a href= "" onClick="this.href=frage();"

                onMouseOver="status=
                    'Ermittlung des richtigen FTP-Servers';"
                onMouseOut = "status = ''">
                Zum FTP-Server
        </a>
</P>
</body>
</html>
```

Führt der Benutzer die Maus auf den Link (onMouseOver), wird in der Statuszeile des Browsers (vgl. Abb. 9.8) der String “Ermittlung des richtigen FTP-Servers” eingeblendet. status ist eine Eigenschaft des window-Objektes (status ist damit äquivalent zu window.status) und kann von einem JavaScript-Programm geändert werden. Verlässt der Mauszeiger den Link (onMouseOut), wird die Anzeige im Statusfeld gelöscht (durch den leeren String ersetzt).

Klickt der Benutzer auf den Link(onClick), wird die Funktion frage() ausgeführt. frage() setzt zunächst den Beginn des Links, auf den später verzweigt werden soll (meinLink = “ftp://www.firma”). Danach wird ein Eingabefenster aufgeblendet, in dem der Benutzer seine Länderkennung eingeben soll. Abhängig von der vom Benutzer eingegebenen Länderkennung, wird dann in einem Switch die URL des dem Benutzer nächstgelegenen FTP-Servers ermittelt.

Die ermittelte URL wird der Eigenschaft href des Linkobjekts zugewiesen. href enthält die Sprungadresse des Links. Das Schlüsselwort `this` bedeutet, dass der gerade aktuelle Link (der Link dessen onClick Event-Handler ausgeführt wurde) gemeint ist (vgl. die Diskussion des Schlüsselworts `this` in Kapitel 8. Auch hier referenziert `this` das Objekt selber). Der Link wird nun ausgeführt und die erste Seite des FTP-Servers in den Browser geladen. Beachten Sie, dass die Sprungadresse des Links

im Tag auf den Leerstring gesetzt wird.

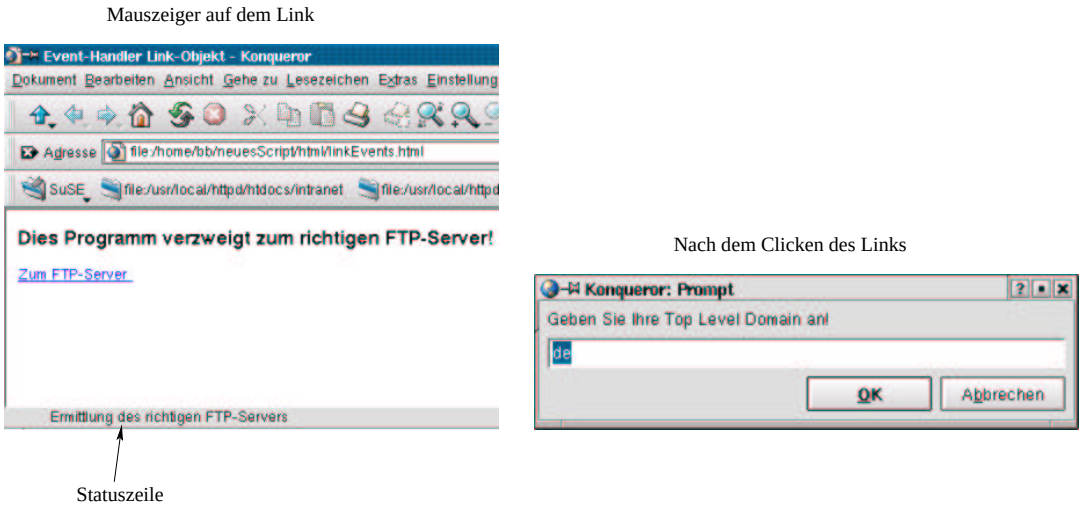


Abbildung 9.8: Event-Handler des Link-Objekts

Ein weiteres bekanntes Beispiel zur Nutzung des Link-Objekts (Mouse-Rollovers) wird im Zusammenhang mit Bildern (image-Objekt) in Kapitel ?? diskutiert.

9.5 Das Form-Objekt

Eine der interessantesten Eigenschaften von JavaScript ist die Zusammenarbeit mit Formularen. JavaScript kann die Werte von Texteingabefeldern und anderen Formularelementen auslesen und ändern.

Formulare werden in html durch das Form-Tag repräsentiert. Das Form-Tag wird in JavaScript auf ein Form-Objekt abgebildet. Form-Objekte, sowie alle ihre Elemente (Textfelder, Buttons, etc) verfügen über Event-Handler, von denen JavaScript-Programme aufgerufen werden können.

Veranschaulichen wir uns dies an einer weiteren Lösung unseres Euro-Dollar-Konvertierungsprogramms:

Beispiel 9.10 *Euro zum Vierten in JavaScript: Event-Handler und Formulare*

```
<!-- Euro-Dm Umrechnung Teil 4
  Dateiname: euro4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 3</title>
  <script language = "JavaScript"
            src="./javascript/euroDollarUmrechnung.js">
  </script>
  <script language = "JavaScript">
    function rechneUm()
    {
        if((document.euro4.zielwaehrung.value=="Dollar")||
            (document.euro4.zielwaehrung.value=="dollar") ||
            (document.euro4.zielwaehrung.value=="euro")||
            (document.euro4.zielwaehrung.value=="Euro"))
        {
```

```
            document.euro4.ergebnis.value=
               euroDollarUmrechnung(
                document.euro4.zielwaehrung.value,
                document.euro4.betrag.value)

        }
        else
        {
            alert("Falsche Zielwährung: \n" +
                                "Erlaubt sind: Euro oder Dollar!");
        }
    }
    </script>

</head>
<body>
    <form name='euro4'>
        <table border>
            <tr>
                <td>
                    Zielw&auml;hrung
                </td>
                <td>
                    <input type="text" name="zielwaehrung" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Betrag
                </td>
                <td>
                    <input type="text" name="betrag" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Ergebnis
                </td>
                <td>
                    <input type="text" name="ergebnis" size=12>
                </td>
            </tr>

            <tr>
                <td colspan="2" align="center">
                    <input type="button"
                        value="Umwandeln" onClick="rechneUm()">
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

In Beispiel 9.10 wird im `<body>` der html-Datei zunächst ein Formular definiert. Das Formular erhält über das name-Attribut den Namen “euro4”. Das form-Objekt ist eine Eigenschaft des document-Objekts. Das in der html-Datei enthaltene Formular ist jetzt vermittels

```
document.euro4
```

von JavaScript aus ansprechbar. In dem Formular selber werden Eingabetextfelder (Elemente des Formulars) definiert. Auch diese Felder erhalten über das name-Attribut Namen. Alle Formular-Elemente (also auch die in Beispiel 9.10 definierten Eingabetextfelder) sind Eigenschaften des form-Objekts (welches selber wieder eine Eigenschaft des document-Objekts war). Daher ist also das Eingabetextfeld mit dem Namen “ergebnis” vermittels

```
document.euro4.ergebnis
```

von JavaScript aus ansprechbar. Die Eingabetextelemente ihrerseits besitzen wieder Eigenschaften (es sind schließlich Objekte). Die wichtigste Eigenschaft der Eingabetextfelder ist die Eigenschaft value. value enthält den derzeitigen Wert des Eingabetextfeldes. Mit der Syntax

```
meineVariable = document.euro4.ergebnis.value;
```

kann also der Wert des Eingabetextelementes “ergebnis” gelesen und mit

```
document.euro4.ergebnis.value = meinWert;
```

geschrieben werden. Auf gleiche Art kann der Wert eines Input-Feldes in Vergleichen auftauchen oder Übergabeparameter einer Funktion sein.

Darüber hinaus enthält das Formular einen Button. Buttons besitzen den Event-Handler onClick, der (natürlich) dann ausgelöst wird, wenn der Benutzer auf den Button klickt. In Beispiel 9.10 ruft der onClick-Event-Handler die Funktion rechneUm() auf:

```
<input type="button" value="Umwandeln" onClick="rechneUm()">
```

Die Funktion rechneUm() prüft nun mit der oben dargestellten Syntax, ob die Zielwährung Dollar oder Euro ist. Ist dies der Fall, wird die in Kapitel 7 entwickelte Funktion zur Euro-Dollar-Umrechnung aufgerufen. Sie erhält als Parameter, wiederum mit der oben beschriebenen Syntax, die jetzigen Werte der Input-Felder zielwaehrung und betrag. Der Rückgabewert der Funktion wird in das Inputtextfeld ergebnis geschrieben:

```
document.euro4.ergebnis.value=
           euroDollarUmrechnung(
            document.euro4.zielwaehrung.value,
            document.euro4.betrag.value)
```

Wurde eine falsche Zielwährung eingegeben, wird der Fehler in einem alert-Fenster dargestellt. Beachten Sie, dass wir hier document.write() nicht benutzen können, da die Darstellung der html-Seite bereits abgeschlossen ist. Vorteilhaft an der Nutzung des alert-Fensters ist Darüber hinaus, dass die html-Seite in der Browser-Darstellung verbleibt. Der Benutzer sieht seinen Fehler, kann diesen korrigieren und das Dokument erneut abschicken. Abb. 9.9 stellt dies noch einmal zusammenfassend dar:

Es gibt allerdings eine weitaus elegantere Lösung der Euro-Dollar-Umrechnung. Auch input-Felder verfügen über Event-Handler. Einer davon ist der Event-Handler onChange. onChange wird

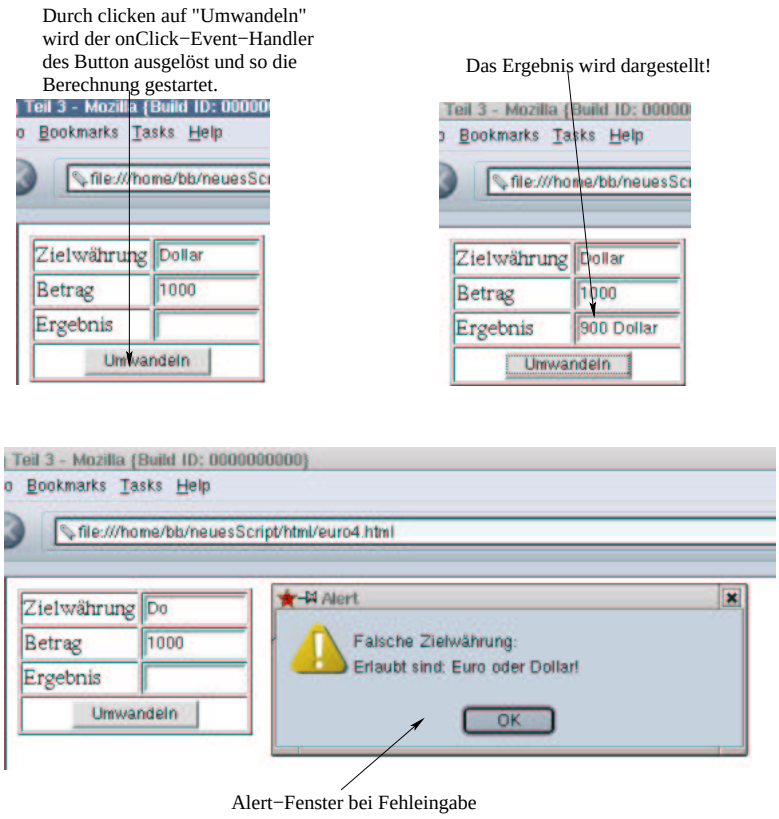


Abbildung 9.9: Euro-Dollar-Umrechnung mit Formular und onClick-Event

ausgelöst, wenn sich der Inhalt eines Input-Text-Feldes ändert. Dies ist immer dann der Fall, wenn der Benutzer den Cursor in einem Input-Text-Feld positioniert, den Inhalt ändert, dann die return-Taste oder die Tab-Taste drückt oder den Cursor durch einen Mausclick aus dem Input-Text-Feld hinausführt:

Beispiel 9.11 *Euro zum Fuenften in JavaScript: Nutzung des onChange Event-Handlers der Input-Text-Felder*

```
<!-- Euro-Dm Umrechnung Teil 5
  Dateiname: euro5.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 5</title>
  <script language = "JavaScript"
            src="./javascript/euroDollarUmrechnung2.js">
  </script>
  <script language = "JavaScript">
    function dollar2euro()
    {
            document.euro5.euro.value=
                        euroDollarUmrechnung(
                            "euro", document.euro5.dollar.value);
    }
```

```
    function euro2dollar()
    {
            document.euro5.dollar.value=
                        euroDollarUmrechnung(
                            "dollar", document.euro5.euro.value);
    }

    </script>
</head>
<body>
    <form name='euro5'>
        <table border>
            <tr>
                <td>
                    Dollar
                </td>
                <td>
                    <input type="text" name="dollar"
                        onChange="dollar2euro()" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Euro
                </td>
                <td>
                    <input type="text" name="euro"
                        onChange="euro2dollar()" size=12>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

In dieser Lösung sind innerhalb des Formulars mit Namen euro5 nur noch zwei Textfelder mit den Namen dollar, resp. euro definiert. Diese Textfelder sind also über

```
document.euro5.dollar
```

bzw.

```
document.euro5.euro
```

ansprechbar. Beide Textfelder sind mit einer Funktion verbunden, die aufgerufen wird, wenn das Textfeld seinen Wert ändert. Die Funktionen entnehmen den aktuellen Wert aus dem geänderten Textfeld, rechnen den Wert mit der in Kapitel 7 programmierten Funktion um und schreiben den umgerechneten Betrag in das jeweils andere Textfeld. Abb. 9.10 stellt dies noch einmal zusammenfasend dar:

9.5.1 Eingabekontrollen

Eine weitere sehr sinnvolle Einsatzmöglichkeit von JavaScript ist die Kontrolle von Benutzereingaben.Ich demonstriere auch dies zunächst am Euro-Dolllar-Umrechnungsbeispiel. Umrechnungen

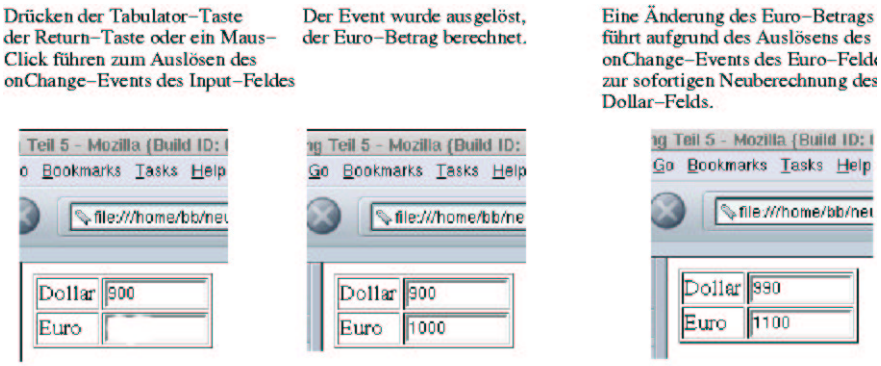


Abbildung 9.10: Euro-Dollar-Umrechnung mit Formular und onChange-Event

machen hier eigentlich nur Sinn, wenn der Benutzer Zahlen[9] eingibt. Eigentlich sollte man diese Fehlermöglichkeit abfangen und den Benutzer auf den Fehler aufmerksam machen.

Eingabekontrollfunktionen lagert man natürlich auch in eigene Dateien aus, schließlich können wir solche Funktionen, wenn wir sie allgemein genug programmieren, von zahlreichen Formularseiten aus nutzen. Beispiel 9.12 zeigt eine Implementierung einer ersten Eingabekontrollfunktion.

Beispiel 9.12 *Eine erste Eingabekontrollfunktion*

```
// Dateiname: feldKontrolle.js

function istKeineZahl (feld, fehlermeldung)
{
if (isNaN(feld.value) ||  (feld.value == ""))
{
feld.focus();
alert(fehlermeldung);
return true;
}
return false;
}
```

Die in Beispiel 9.12 dargestellte Funktion istKeineZahl() erwartet zwei Übergabeparameter. Der erste Übergabeparameter muss ein Input-Textfeld sein, der zweite ein String. In der Zeile

```
if (isNaN(feld.value) ||  (feld.value == ""))
```

wird zunächst die JavaScript interne Methode isNaN[10] angewendet. Diese Funktion prüft, ob die ihr übergebene Variable zur Zeit eine Zahl enthält. Leider ist für isNaN aber auch der Leerstring (“”) eine

[9] Buchstaben und Sonderzeichen lassen sich extrem schlecht umrechnen!

[10] Abkürzung für is Not a Number

Zahl[11]. Aus diesem Grund testen wir in einer Oder-Verknüpfung, ob der Leerstring übergeben wurde. Der Ausdruck

```
feld.value
```

macht nur dann Sinn, wenn der erste Übergabeparameter ein Textfeld ist. Dann nämlich ist value eine Eigenschaft des Textfeldes, nämlich der jetzige Inhalt dieses Feldes. Hier sehen wir ein weiteres Mal, dass jedes Objekt in der Übergabeparameterliste einer Funktion stehen darf.

Ist der Inhalt des übergebenen Feldes keine Zahl, werden die Anweisungen im `if`-Block ausgeführt. Hier sorgt

```
feld.focus();
```

zunächst dafür, dass der Cursor in das fehlerhaft ausgefüllte Textfeld springt. focus() ist eine Methode des Textfeld-Objekts und plaziert den Cursor in eben jenes Textfeld[12]. Danach wird die als zweiter Parameter übergebene Fehlermeldung in einem alert-Fenster ausgegeben (vgl. Abb. 9.11). Die Funktion beendet sich, indem sie true zurückgibt.

Ist der derzeitige Wert des übergebenen Textfeldes hingegen eine Zahl, gibt die Funktion false zurück und beendet sich[13].

Nun müssen wir unsere Funktion in das Euro-Dollar-Umrechnungsbeispiel einbauen. Dies zeigt Beispiel 9.13.

Beispiel 9.13 *Euro zum Sechsten in JavaScript: Eingabekontrolle*

```
<!-- Euro-Dm Umrechnung Teil 5
  Dateiname: euro6.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Euro-Dollar Umrechnung Teil 6</title>
  <script language = "JavaScript"
            src="./javascript/euroDollarUmrechnung2.js">
  </script>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript">
    function dollar2euro()
    {
            if(istKeineZahl(document.euro6.dollar,
                              "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            {

            document.euro6.euro.value=
                        euroDollarUmrechnung(
```

[11] Was der definitiv nicht ist!

[12] gibt dem Textfeld den Fokus

[13] Beachten Sie die Umkehrung der Logik: Die Funktion heisst istKeineZahl. Damit ist sie wahr, wenn das übergebene Feld keine Zahl enthält, und falsch, wenn sie eine Zahl enthält!

```
                            "euro", document.euro6.dollar.value);
            }
    }
    function euro2dollar()
    {
            if(istKeineZahl(document.euro6.euro,
                                "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            document.euro6.dollar.value=
                        euroDollarUmrechnung(
                            "dollar", document.euro6.euro.value);
    }

    </script>
</head>
<body>
    <form name='euro6'>
        <table border>
            <tr>
                <td>
                    Dollar
                </td>
                <td>
                    <input type="text" name="dollar"
                        onChange="dollar2euro()" size=12>
                </td>
            </tr>
            <tr>
                <td>
                    Euro
                </td>
                <td>
                    <input type="text" name="euro"
                        onChange="euro2dollar()" size=12>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Hier wird unsere Eingabekontrollfunktion zunächst in das Umrechnungsprogramm geladen. Die Datei, die sie enthält, heisst feldkontrolle.js und liegt, wie gewohnt, im Verzeichnis javascript unterhalb des Verzeichnisses, in dem unsere Umrechnungsdatei beheimatet ist:

```
    <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
    </script>
```

Die einzigen weiteren Änderungen betreffen die Funktionen, die über die onChange-Event-Handler der Textfelder gestartet werden. Hier wird jeweils als erstes überprüft, ob der Inhalt des geänderten Textfeldes eine Zahl ist:

```
if(istKeineZahl(document.euro6.dollar,
                              "Keine Zahl eingegeben!"))
```

Der erste Übergabeparameter ist ein Textfeld-Objekt[14]. Also macht es in istKeineZahl auch Sinn, mit feld.value[15] den Inhalt dieses Feldes zu ermitteln. Der zweite Übergabeparameter ist der String, der im Fehlerfall ausgegeben werden soll. Ist der Inhalt des zu überprüfenden Feldes keine Zahl, beendet sich auch die über den Event-Handler gestartete Funktion und nichts passiert (ausser, dass das alert-Fenster mit der Fehlermeldung erscheint.). Im anderen Fall wird die Berechnung durchgeführt.

Darüber hinaus sehen wir hier die Aufsplittung der Logik in eine allgemeine Funktion (istKeineZahl()), die in einer eigenen Datei abgespeichert wird und so für jedes Formular zur Verfügung steht und den über den Event-Handler aufgerufenen Funktionen. dollar2euro resp. euro2dollar machen ohne das spezielle Formular keinen Sinn, istKeineZahl() hingegen schon.

Abb. 9.11 zeigt eine Fehlerausgabe dieses Programms.

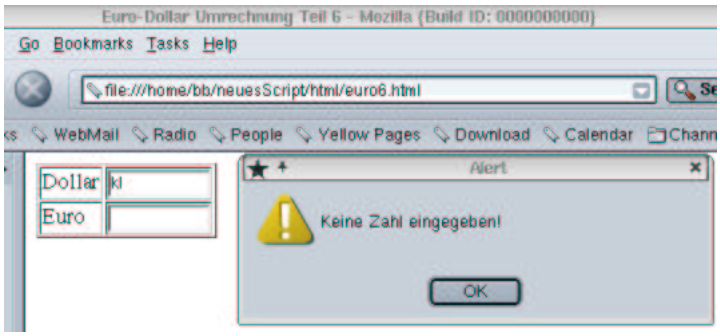


Abbildung 9.11: Euro-Dollar-Umrechnung mit Formular und Eingabekontrolle

Ihre wahre Stärke spielt Eingabekontrolle in JavaScript aber in Zusammenarbeit mit serverseitigen Anwendungen aus. Betrachten wir hier als einfaches Beispiel ein Online-Bestellformular wie in Abb. 9.12.

Eine reine JavaScript-Lösung scheidet hier aus. Die Eingaben der Benutzer müssen ja irgendwie zum Server, um da in eine Datenbank geschrieben zu werden, oder an ein BWL-Software-System übergeben zu werden. Also bietet sich eine php-Lösung[16] an. Allerdings müssen die Eingaben der Benutzer überprüft werden:

- Name, Vorname, Stadt, Strasse, und Artikelname müssen gefüllt sein.
- PLZ, Artikelnummer und Anzahl müssen Zahlen sein.

Wir könnten die Eingabeprüfungen auf dem Server mit php durchführen. Dies ist aber keine optimale Lösung:

- Wenn Eingabefehler auftreten, ist es eigentlich sinnlos, die Eingaben an den Server zu übertragen. Sie können dort ohnehin nicht verarbeitet werden.

[14] Das Textfeld mit dem Namen dollar im Formular mit dem Namen euro6 im document-Objekt.
[15] document.euro6.dollar wird ja auf feld übergeben.
[16] Auch wenn Sie noch nicht wissen, wie man aus php in eine Datenbank schreibt.

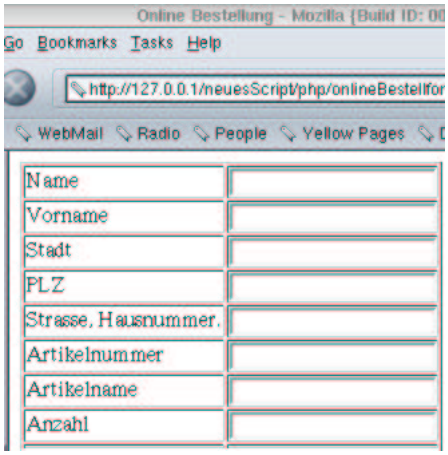


Abbildung 9.12: Ein einfaches Onlinebestellformular

- Das Eingabeformular muss beim Benutzer wieder aufgebaut werden. Seine fehlerfreien Eingaben sollten bei der erneuten Darstellung noch vorhanden sein. Dies erhöht den Programmieraufwand.
- Dem Benutzer muss klar gesagt werden, was er tun muss, um den Fehler beim erneuten Ausfüllen des Formulars zu vermeiden. Dazu sollte der Cursor im Textfeld mit der Fehleingabe stehen. Die Anleitung zur Behebung des Fehlers sollte deutlich sichtbar an herausgehobener Stelle positioniert sein. Dies ist ohne JavaScript schwer möglich.

Das bedeutet, dass JavaScript auch im Zusammenhang mit Formularen, die an den Server übertragen werden sollen, interessant ist. Eine mögliche Lösung der oben beschriebenen Problematik ist die Nutzung eines Event-Handlers des submit-Buttons. Ebenso, wie die in Beispiel 9.10 genutzten einfachen Buttons, besitzen auch die Submit-Buttons den Event-Handler onClick. Gibt onClick true zurück, wird das Formular übertragen, gibt onClick false zurück, wird das Klicken auf den Submit-Button ignoriert. Dadurch können auf einfache Weise Eingabeüberprüfungen vorgenommen werden. Wenden wir uns nun einer Lösung der in Abb. 9.12 dargestellten Problemstellung zu. Als erstes müssen wir eine neue Prüfroutine schreiben. Denn bislang haben wir nur Funktionen, die testen, ob eine Eingabe eine Zahl ist. Hier gibt es Eingaben, die vorhanden sein müssen, aber nicht unbedingt Zahlen sein müssen. Beispiel 9.14 zeigt die erweitereten Prüfroutinen:

Beispiel 9.14 *Erweiterung der Prüffunktionen*

```
// Dateiname: feldKontrolle.js

function istKeineZahl (feld, fehlermeldung)
{
    if (isNaN(feld.value) ||  (feld.value == ""))
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
```

```
function istLeer (feld, fehlermeldung)
{
    if (feld.value == "")
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
```

Hier ist die Funktion istLeer() hinzugekommen. Genau wie istKeineZahl() erwartet istLeer() zwei Übergabeparameter: Das zu überprüfende Feld und die Meldung, die im Fehlerfall ausgegeben werden soll. Die Implementierung der Funktion ist vollständig analog zu istKeineZahl(). Eine mögliche Implementierung des Online-Formulars zeigt nun Beispiel 9.15.

Beispiel 9.15 *Erste Implementierung des Online-Formulars*

```
<!-- Online Bestellformular Teil1
  Dateiname: onlineBestellformular1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Online Bestellung</title>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle2.js">
  </script>
  <script language = "JavaScript">
    function teste()
    {
        if(istLeer(document.bestellung.name,
            "Name nicht ausgefüllt!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.vorname,
            "Vorname nicht ausgefüllt!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.stadt,
            "Stadt nicht ausgefüllt!"))
        {
            return false;
        }
        if(istKeineZahl(document.bestellung.plz,
            "PLZ ist keine Zahl!"))
        {
            return false;
        }
```

```
        if(istLeer(document.bestellung.strasse,
            "Strasse nicht ausgefüllt!"))
        {
            return false;
        }
        if(istKeineZahl(document.bestellung.artikelnummer,
            "Artikelnummer ist keine Zahl!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.artikelname,
            "Name nicht ausgefüllt!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.anzahl,
            "Anzahl ist keine Zahl!"))
        {
            return false;
        }
        return true;
    }
  </script>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='bestellung' action='$PHP_SELF' method='post'>";
?>
            <table border="1">
                <tr>
                    <td> Name </td>
                    <td>
                        <input type="text"
                            name="name" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Vorname </td>
                    <td>
                        <input type="text"
                            name="vorname" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Stadt</td>
                    <td>
                        <input type="text"
                            name="stadt" size="25">
                    </td>
                </tr>
```

```
                <tr>
                    <td> PLZ </td>
                    <td> <input type="text"
                        name="plz" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Strasse, Hausnummer.</td>
                    <td>
                        <input type="text"
                            name="strasse" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Artikelnummer </td>
                    <td>
                        <input type="text"
                            name="artikelnummer" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Artikelname </td>
                    <td>
                        <input type="text"
                            name="artikelname" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Anzahl </td>
                    <td>
                        <input type="text"
                            name="anzahl" size="25">
                    </td>
                </tr>
                <tr>
                    <td align="center">
                        <input type="submit" onClick="return(teste())"
                            value="Abschicken">
                    </td>
                    <td align="center">
                        <input type="reset"
                            onClick='return(confirm("Wirklich löschen?"))'
                            value="L&ouml;schen">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        /*
            Code, um die Bestellung in die Datenbank zu schreiben.
            Koennen wir noch nicht --> Kapitel ??
```

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

In Beispiel 9.15 werden im <head>-Teil zunächst die Eingabekontrollfunktionen geladen:

```
<script language = "JavaScript"
          src="./javascript/feldkontrolle2.js">
</script>
```

JavaScript ist also auch in php-Seiten erlaubt[17]. Im Anschluss wird die JavaScript-Funktion definiert, die die Fehlerkontrollfunktionen aufruft und entscheidet, welche dieser Funktionen auf welches Feld angewendet wird.

Im <body> wird zunächst, wie immer bei Formular-Seiten mit php, überprüft, ob die Datei vom Browser mit get (erster Aufruf, Formular muss aufgebaut werden) oder post (zweiter Aufruf, Eingaben des Formulars müssen verarbeitet werden) angefordert wird. Erfolgte der Aufruf über get, wird das Formular aufgebaut, über das die Artikel bestellt werden sollen. Der Benutzer muss die Eingaben bzgl. Name, Adresse und Artikel vornehmen und kann danach vermittels Klicken auf den Submit-Button den Inhalt des Formulars an den Server übersenden, welcher die Eingaben dann weiterverarbeiten kann.

Bevor das Formular übertragen wird, wird vermittels des onClick-Event-Handlers der Rückgabewert der Funktion teste() an das form-Objekt übergeben. teste() überprüft, ob in den Feldern PLZ, Artikelnummer und Anzahl eine Zahl steht und ob die Felder für Name, Vorname, Stadt, Strasse und Artikelname nicht leer sind.

Dazu bedient sich teste() unserer Eingabekontrollfunktionen istKeineZahl() und istLeer(). Kommt von istKeineZahl() oder istLeer() bei einer Überprüfung true zurück, gibt teste() false zurück, das Formular wird nicht übertragen, der Benutzer sieht die Fehlermeldung. Das betreffende Feld hat bereits den Fokus (durch istKeineZahl() oder istLeer()) und der Benutzer kann seine Eingaben korrigieren.

Antwortet istLeer() und istKeineZahl() immer mit false, gibt teste() true zurück und das Formular wird übertragen.

Auch hier sehen wir wieder die Aufsplittung der Logik in zwei allgemeine Funktionen (istKeineZahl() und istLeer()), die in einer eigenen Datei abgespeichert werden und so für jedes Formular zur Verfügung stehen und der Funktion teste(). teste() macht ohne das spezielle zu überprüfende Formular keinen Sinn, da teste() ja den Namen des Formulars und der Eingabefelder nutzt.

Beispiel 9.15 zeigt Darüber hinaus eine Nutzung des onClick-Event-Handlers des Reset-Buttons. Klicken des Reset-Buttons setzt alle Elemente des Formulars auf den Leerstring. Insbesondere bei großen Formularen macht es Sinn, noch einmal nachzufragen, ob der Benutzer auch wirklich alle seine Eingaben löschen will. Der onClick-Event-Handler des Reset-Buttons funktioniert völlig analog zum onClick-Event-Handler des Submit-Buttons. Gibt er true zurück, wird die Aktion durchgeführt, ansonsten nicht. In Beispiel 9.15 blenden wir nach dem Klicken auf den Reset- Button ein Confirm-Fenster auf. Klickt der Benutzer hier auf OK, gibt confirm() true zurück, die Inhalte der Felder werden gelöscht. Klickt der Benutzer hingegen auf Cancel, wird die Aktion abgebrochen und alle Eingaben bleiben erhalten (s. Abb. 9.13).

[17] Was an sich ganz klar ist, denn nur Teile der Seite, die zwischen <?php und ?> stehen, werden dem php-Modul übergeben, der Rest wird ungeändert an den Browser übertragen und der kann (hoffentlich) JavaScript.

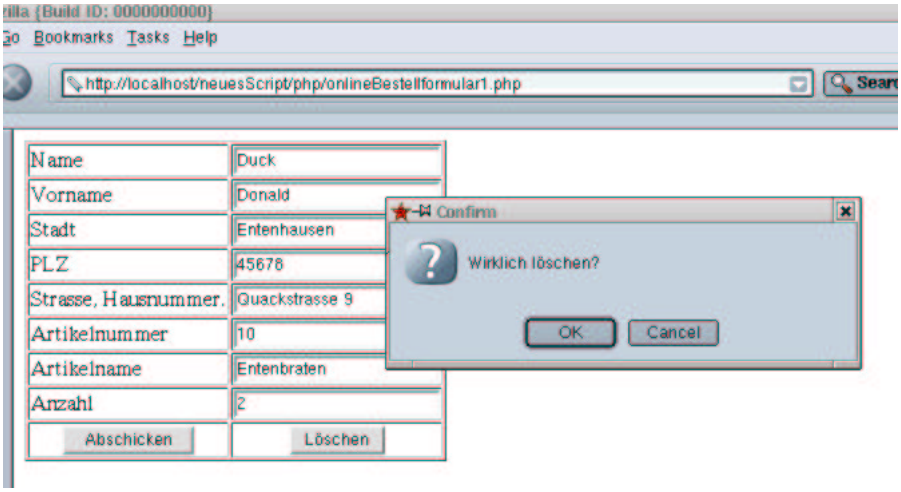


Abbildung 9.13: Das Onlinebestellformular nach Betätigung des Reset-Button

Die Logik auf Serverseite wurde noch nicht implementiert, weil wir das noch nicht können.

Beispiel 9.15 hat einen kleinen Schönheitsfehler: JavaScript kann man im Browser abschalten. Abb. 9.14 zeigt dies am Beispiel des Mozilla-Browsers.

Auch im Internet Explorer kann man JavaScript abschalten (obwohl das im IE mit seinen Millionen von Einstellmöglichkeiten nicht wirklich einfach zu finden ist).

Wenn Benutzer aber JavaScript abschalten, haben wir ein Problem mit unserer Lösung. Der Submit-Button funktioniert weiterhin, JavaScript hin oder her. Was nicht läuft, sind die Eingabekontrollroutinen. Das bedeutet, es kann Datenmüll auf dem Server ankommen. Wir müssten auf dem Server in php erneut prüfen, ob die Eingaben in Ordnung sind. Dies ist aber, wie oben bereits dargestellt, erheblicher Aufwand. Es gibt aber eine einfache Möglichkeit, auch dieses Problem zu lösen: Das form-Objekt in JavaScript verfügt über die Methode submit(). Auch damit kann ein Formular abgeschickt werden. Dies funktioniert aber nur dann, wenn JavaScript eingeschaltet ist. Wir könnten also das Abschicken des Formulars mit der submit-Methode in die Funktion teste() aufnehmen. Um zu verhindern, dass das Formular abgeschickt werden kann, wenn der Benutzer kein JavaScript angeschaltet hat, verzichten wir auf den Submit-Button. Wir benutzen einfach einen normalen Button, binden über den onClick-Event-Handler normaler Buttons die Funktion teste() an den Button und verschicken das Formular aus der Funktion. Dann kann eigentlich nichts mehr passieren, da das Formular nur dann abgeschickt werden kann, wenn der Benutzer JavaScript eingeschaltet hat. Dann sind aber auch die Tests durchgeführt. Beispiel 9.16 zeigt die Implementierung:

Beispiel 9.16 *Sicherere Implementierung des Online-Formulars*

```
<!-- Online Bestellformular Teil2
  Dateiname: onlineBestellformular2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Online Bestellung</title>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle2.js">
  </script>
  <script language = "JavaScript">
```

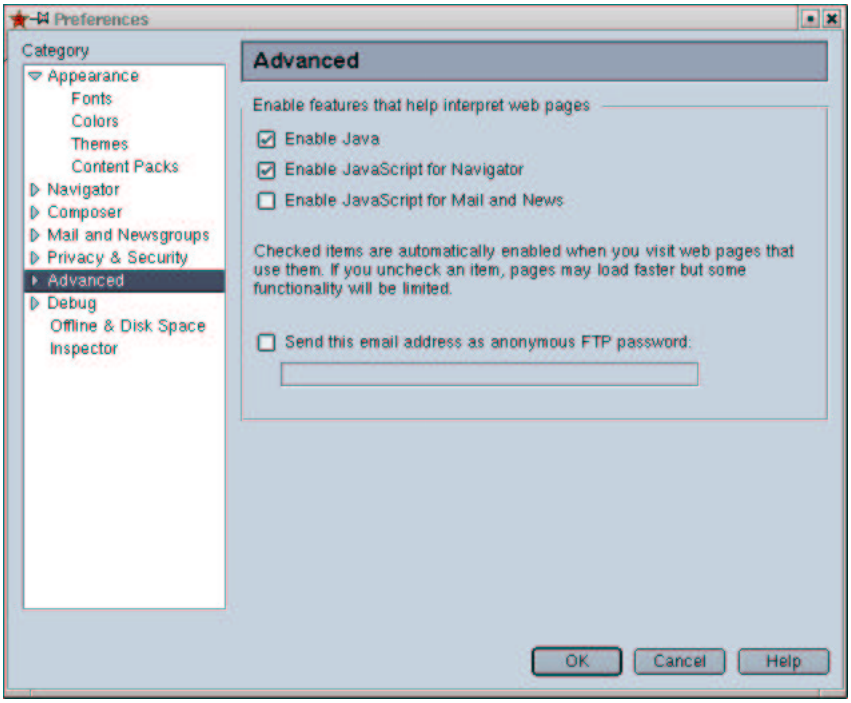


Abbildung 9.14: JavaScript abschalten in Mozilla oder Netscape

```
function teste()
{
    if(istLeer(document.bestellung.name,
        "Name nicht ausgefüllt!"))
    {
        return false;
    }
    if(istLeer(document.bestellung.vorname,
        "Vorname nicht ausgefüllt!"))
    {
        return false;
    }
    if(istLeer(document.bestellung.stadt,
        "Stadt nicht ausgefüllt!"))
    {
        return false;
    }
    if(istKeineZahl(document.bestellung.plz,
        "PLZ ist keine Zahl!"))
    {
        return false;
    }
    if(istLeer(document.bestellung.strasse,
        "Strasse nicht ausgefüllt!"))
    {
        return false;
    }
```

```
        if(istKeineZahl(document.bestellung.artikelnummer,
            "Artikelnummer ist keine Zahl!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.artikelname,
            "Name nicht ausgefüllt!"))
        {
            return false;
        }
        if(istLeer(document.bestellung.anzahl,
            "Anzahl ist keine Zahl!"))
        {
            return false;
        }

        document.bestellung.method="post";
        document.bestellung.action=document.bestellung.anWen.value;
        document.bestellung.submit();
    }
  </script>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
?>
        <form name='bestellung'>
            <input type="hidden" name="anWen"
                        value="<?php echo $PHP_SELF ?>" >
            <table border="1">
                <tr>
                    <td> Name </td>
                    <td>
                        <input type="text"
                            name="name" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Vorname </td>
                    <td>
                        <input type="text"
                            name="vorname" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Stadt</td>
                    <td>
                        <input type="text"
                            name="stadt" size="25">
                    </td>
                </tr>
```

```
                <tr>
                    <td> PLZ </td>
                    <td> <input type="text"
                        name="plz" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Strasse, Hausnummer.</td>
                    <td>
                        <input type="text"
                            name="strasse" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Artikelnummer </td>
                    <td>
                        <input type="text"
                            name="artikelnummer" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Artikelname </td>
                    <td>
                        <input type="text"
                            name="artikelname" size="25">
                    </td>
                </tr>
                <tr>
                    <td> Anzahl </td>
                    <td>
                        <input type="text"
                            name="anzahl" size="25">
                    </td>
                </tr>
                <tr>
                    <td align="center">
                        <input type="button" onClick="teste()"
                            value="Abschicken">
                    </td>
                    <td align="center">
                        <input type="reset"
                            onClick='return(confirm("Wirklich löschen?"))'
                            value="L&ouml;schen">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        /*
            Code, um die Bestellung in die Datenbank zu schreiben.
            Koennen wir noch nicht --> Kapitel ??
```

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

Betrachten wir zunächst die Änderungen im html. Wir beginnen bei der Definition des Formulars. Das `<form>`-Tag wird zu:

```
<form name='bestellung'>
```

Hier ist keine Rede mehr von Parametern, wie action oder method. Dies brauchen wir auch nicht mehr, weil das Formular ja über JavaScript und nicht aus html versendet wird. Neu ist die Zeile:

```
<input type="hidden" name="anWen"
            value="<?php echo $PHP_SELF ?>" >
```

Diese Zeile erstellt ein verstecktes Inputfeld (type="hidden"). Das Feld erhält als Wert den Namen der php-Seite selber (echo $PHP_SELF). Den Sinn können Sie sicher[18] erraten. Wir müssen ja irgendwo noch die Aktion angeben, die auf dem Server ausgeführt werden soll, wenn die Eingaben übermittelt werden (der Ersatz des action-Parameters im `<form>-Tag`). Das geschieht über den Wert dieses Feldes. Die letzte html-Änderung macht aus dem Submit-Button einen normalen Button:

```
<input type="button" onClick="teste()"
                        value="Abschicken">
```

Die Funktionalität, das Formular versenden zu können, ist jetzt aus dem html-Code entfernt. In JavaScript müssen wir diese Funktionalität nun reimplementieren. Die einzige Änderung in JavaScript betrifft die Funktion teste(). Hier wird `return` true durch folgende Zeilen ersetzt:

```
document.bestellung.method="post";
document.bestellung.action=document.bestellung.anWen.value;
document.bestellung.submit();
```

Die erste Zeile zeigt uns die method-Eigenschaft des form-Objekts in JavaScript. Diese Eigenschaft erhält den Wert "post". Der method-Eigenschaft des form-Objekts einen Wert zuzuweisen entspricht dem Setzen des method-Parameters im html `<form>`-Tag. Die zweite Zeile setzt die action-Eigenschaft des JavaScript form-Objekts auf den Wert des anWen Textfeldes. Dieser Wert ist aber der Name der php-Seite selber. Und dies entspricht dem Setzen des action-Parameters im html `<form>`-Tag. Die letzte Zeile schickt das Formular an den Server.

Hat der Benutzer JavaScript abgeschaltet, passiert gar nichts.

Aufgabe 9.1 *In Beispiel 9.16 wird das Feld für Postleitzahlen überprüft. Die Überprüfung beschränkt sich aber darauf, zu testen, ob in diesem Feld eine Zahl enthalten ist. Schreiben Sie eine Funktion, die testet, ob es diese Zahl wirklich eine Postleitzahl sein kann.*

Aufgabe 9.2 *Sie sollen für eine Bank die Errechnung von Darlehenskonditionen für Kunden der Bank über das Internet ermöglichen. Eingegeben werden soll das Eigenkapital, der Preis der Immobilie, die gekauft werden soll, der Zinssatz und die Tilgung. Das Programm soll die monatliche Belastung*

[18] oder vielleicht auch nicht?

ausgeben. Wenn die Eigenkapitalquote des Kunden kleiner als 30 % ist, soll keine Berechnung durchgeführt werden und anstelle dessen ausgegeben werden, dass die Bank Immobilienerwerb mit einer so geringen Eigenkapitalquote nicht finanziert.

Dabei sollen folgende Nebenbedingungen eingehalten werden: Alle Eingaben müssen positive Zahlen sein. Das Eigenkapital muss kleiner als der Preis der Wohnung sein. Ist eine dieser Bedingungen verletzt, soll der Benutzer darauf hingewiesen werden und es wird keine Berechnung durchgeführt. Diese Prüfungen und die Berechnungen müssen in eigene Funktionen ausgelagert sein.

Darüber hinaus werden folgende Plausibilitätskontrollen durchgeführt: Ist der eingegebene Zinssatz kleiner als 5 %, wird der Benutzer darauf hingewiesen, dass der von ihm eingegebene Zins unterhalb des Zinssatzes der Bank liegt, eine Berechnung wird dennoch durchgeführt. Ist die gewünschte Tilgung größer als 4 %, wird der Benutzer darauf hingewiesen, dass dies ungewöhnlich ist, die Berechnung wird dennoch durchgeführt.

Lösen Sie die Aufgabe zum Einen rein in JavaScript unter Ausnutzung des onClick-Event-Handlers des Button-Objekts. Die Anwendung soll sich, wie in Abb. 9.17 dargestellt, präsentieren.

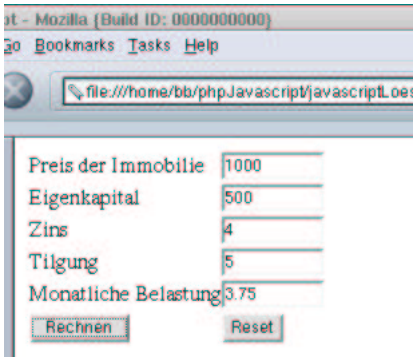


Abbildung 9.15: Bildschirmdarstellung von Aufgabe 9.2

Die zweite Lösung soll die Eingabekontrollen in JavaScript durchführen, die Eingaben dann an den Server übertragen und die Berechnungen danach in php durchführen. Auch hier sollen die Berechnungen in eigene php-Funktionen ausgelagert sein. Dabei soll sichergestellt werden, dass die Eingaben nur dann übertragen werden, wenn die Eingabekontrollen auch durchgeführt wurden. Die Bildschirmdarstellung soll dann wie in Abb. 9.16 sein.

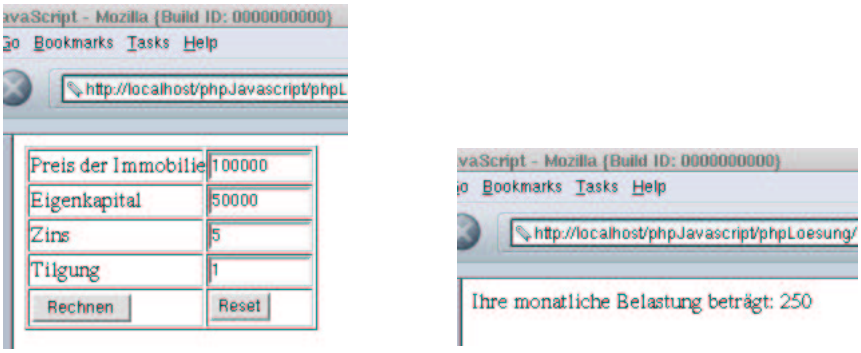


Abbildung 9.16: Bildschirmdarstellung von Aufgabe 9.2

Aufgabe 9.3 *Aufgabe 7.2 soll ebenfalls erweitert werden. Die Anwendung soll sich, wie in Abb.* **??** *dargestellt, präsentieren.*

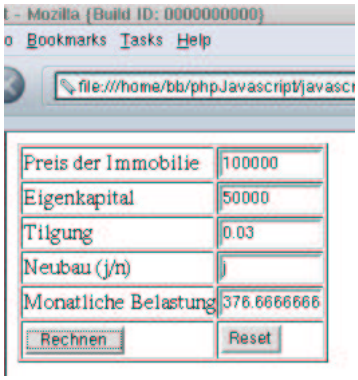


Abbildung 9.17: Bildschirmdarstellung von Aufgabe 9.3

Hierbei sollen folgende Eingabeprüfungen durchgeführt werden:

- *Eigenkapital, Preis und Tilgung müssen positive Zahlen sein.*
- *In das Feld Neubau darf nur "j" oder "n" eingegeben werden.*
- *Der Preis muss höher als das Eigenkapital sein.*

Ist eine dieser Bedingungen verletzt, soll der Benutzer darauf hingewiesen werden und es wird keine Berechnung durchgeführt. Diese Prüfungen und die Berechnungen müssen in eigene Funktionen ausgelagert sein.

Darüber hinaus wird folgende Plausibilitätskontrolle durchgeführt: Ist die gewünschte Tilgung größer als 4 %, wird der Benutzer darauf hingewiesen, dass dies ungewöhnlich ist, die Berechnung wird dennoch durchgeführt.

Lösen Sie die Aufgabe zum Einen in rein in JavaScript unter Ausnutzung des onClick-Event-Handlers des Button-Objekts.

Die zweite Lösung soll die Eingabekontrollen in JavaScript durchführen, die Eingaben dann an den Server übertragen und die Berechnungen dann in php durchführen. Auch hier sollen die Berechnungen in eigene php-Funktionen ausgelagert sein. Dabei soll sichergestellt werden, dass die Eingaben nur dann übertragen werden, wenn die Eingabekontrollen auch durchgeführt wurden. Die Bildschirmdarstellung soll dann wie in Abb. 9.18 sein.

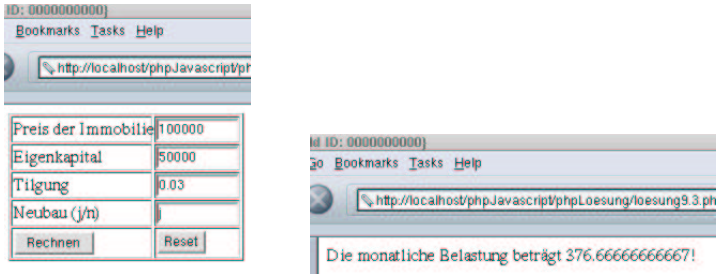


Abbildung 9.18: Bildschirmdarstellung von Aufgabe 9.3

Die Zinsstaffelung kann Aufgabe 7.2 entnommen werden.

Kapitel 10

Das große Ganze: Teil 1

In diesem Kapitel wollen wir das bisher Gelernte am Euro-Dollar-Umrechnungsbeispiel vertiefen. Darüber hinaus werden Sie einige neue Sachen, wie die Behandlung von Strings oder Datenbankzugriff aus php, lernen. Als Datenbank setzen wir MySql ein. MySql ist ein unter der GPL stehendes Datenbanksystem und gerade bei Internet-Anwendungen in Verbindung mit php weit verbreitet. Als Werkzeug zur Administration der Datenbank benutzen wir das ebenfalls unter der GPL stehende phpmyadmin.

10.1 Aufgabenstellung und Analyse

Wir setzen das Euro-Dollar-Umrechnungsbeispiel fort: Es soll ernst werden. Die Euro-Dollar-Umrechnung soll in den Internet-Auftritt der Schwanen Gesellschaft GmbH[1] eingebunden werden. Auf der Produktseite soll, wie in Kapitel 7 erstmals behandelt, der Preis unserer Produkte in Euro und Dollar ausgewiesen werden. Um die Anziehungskraft der Produktseite zu erhöhen und damit für Internet-Benutzer einen zusätzlichen Anreiz zu schaffen, diese Seite zu besuchen, soll im "oberen Bereich"[2] der Seite ein Euro-Dollar-Umrechner angeboten werden. Die Anwendung soll leicht erweiterbar sein. Das bedeutet, wenn unser Marketing auf die Idee kommt, die Preise auch in Schweizer Franken oder in japanischen Yen auszuweisen, soll auch das möglich sein. Der Euro-Dollar-Umrechner muss dann selbstverständlich zu einen Euro-Dollar-Franken- bzw. Euro-Dollar-Yen-Umrechner erweitert werden. Der gegenwärtige Dollarkurs soll von Sachbearbeitern der Produktpflege in das System eingegeben werden. Dies soll jederzeit möglich sein. Der eingegebene Kurs liegt dann allen Berechnungen zu Grunde. Die Sachbearbeiter der Produktpflege sind allerdings alle Computer-Laien. Von ihnen kann nicht erwartet werden, Quellcode zu editieren. Für sie muss daher eine einfache Möglichkeit zur Kurspflege vorgesehen werden. Da wir zur Zeit nur drei Produkte über das Internet anbieten wollen, ist nicht daran gedacht, die Produktpreise aus der vorhandenen Produktdatenbank einzulesen[3]. Für sämtliche Eingaben sollen sowohl Komma als auch Punkt als Dezimaltrenner vorgesehen werden. Die Ausgabe erfolgt immer mit zwei Nachkommastellen und dem Komma als Dezimaltrenner.

Zunächst schreiben wir die Anforderungen (Anwendungsfälle) etwas strukturierter auf:

1. Der Dollarkurs muss über eine einfach zu bedienende Oberfläche eingegeben werden. Als Dezimaltrenner sind Komma und Punkt erlaubt. Die Eingaben werden von Sachbearbeitern der Produktpflege vorgenommen.

[1] Wegen des Logos, werden Sie später sehen!
[2] Was auch immer damit gemeint sein soll!
[3] Das machen wir in Kapitel ??.

2. Dieser Dollarkurs liegt allen Ausgaben zu Grunde.
3. Die Euro-Preise unserer Produkte werden automatisch in Dollar umgerechnet. Diese Preise werden zusätzlich auf der Produktseite ausgegeben.
4. In die Produktseite wird ein Euro-Dollar-Umrechner integriert. Bei den Eingaben sind Punkt oder Komma als Dezimaltrenner erlaubt, alle Ausgaben erfolgen mit zwei Nachkommastellen und dem Komma als Dezimaltrenner.
5. Die Anwendung muss auf weitere Währungen erweiterbar sein.

Als Nächstes erstellen wir ein Use-Case-Diagramm. Use-Case-Diagramme zeigen die Anwendungsfälle (engl. Use Case) und die Anwender (Akteure) die mit den Anwendungsfällen interagieren[4]:

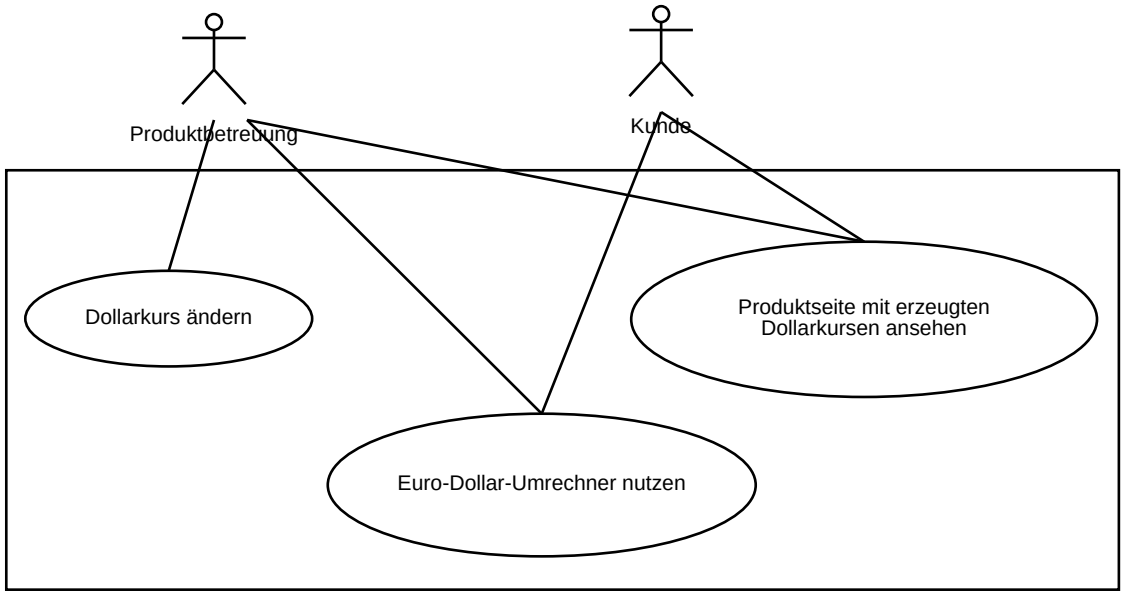


Abbildung 10.1: Das Use-Case-Diagramm zur Euro-Dollar-Problemstellung

Abb. 10.19 sollte selbsterklärend sein.
Aus der Aufgabenstellung leiten wir zunächst folgende Entscheidungen ab:

- Wir werden zwei html-Seiten benötigen:
  - Eine für die Produktbetreuung zur Pflege des Dollarkurses.
  - Eine für die Kunden, mit Produktpreisen und dem Rechner.
- Wir werden den Dollarkurs in einer Datenbank vorhalten. Dazu werden wir eine Tabelle Währung anlegen und als zunächst einzigen Eintrag den Dollarkurs vornehmen. Dies ist flexibel für mögliche spätere Erweiterungen (weitere Währungen). Außerdem werden wir irgendwann auch die Produktpreise aus der Datenbank einlesen, und müssen dann sowieso mit der Datenbank arbeiten. Alternative wäre, den Kurs in einer Datei im Dateisystem des www-Servers abzuspeichern und das ist genauso aufwändig, wie die Nutzung einer Datenbank.

[4]Das Use-Case-Diagramm ist ebenfalls Bestandteil der bereits in Kapitel 8 erwähnten UML.

10.2 Einige Bemerkungen zur Nutzung von Datenbanksystemen in php

Datenbanksysteme sind Serversysteme, die ähnlich wie www-Server auf clientseitige Anforderungen warten. Genau wie die Browser von www-Servern html-Seiten anfordern, können unsere php-Programme von Datenbankservern Daten anfordern. Genau wie bei www-Servern muss man dazu den Namen des Rechners, auf dem die Datenbanksoftware läuft (das kann natürlich auch der www-Server selber sein) wissen. Anders als bei www-Server-Software muss man sich bei Datenbanken mit Benutzernamen und Passwort legitimieren.

Wenn man das tut, erhält man eine bidirektionale Verbindung zur Datenbank. Über diese Verbindung kann man Abfragen und Kommandos an die Datenbank schicken. Die Datenbank antwortet über diese Verbindung. Dies ist dann entweder das Ergebnis der Abfrage oder eine Bestätigung, das Kommando durchgeführt zu haben[5].

Die Abfragen oder Kommandos müssen in SQL (Structured Query Language)[6] formuliert sein.

Ich zeige die grundsätzliche Vorgehensweise an einem Zugriff von php auf ein MySql-Datenbanksystem. Der Verbindungsaufbau zu einer Datenbank erfolgt mit der php-Funktion mysql_pconnect:

```
$link=mysql_pconnect("localhost", "bb", "meinPW");
```

Wie Sie sehen, erwartet mysql-pconnect drei Übergabeparameter:

- Den Namen des Rechners: Dies ist hier localhost. Hiermit wird der Rechner, auf dem das php-Programm selber läuft, angesprochen. Alternativ kann hier der Name eines anderen Rechners stehen. Der Verbindungsaufbau über das Netz erfolgt automatisch.
- Den Namen des Benutzers: Dies ist hier bb. Hier kann der Name eines jeden beim Datenbanksystem registrierten Nutzers stehen.
- Das Passwort des Benutzers: Hier meinPW.

Im Erfolgsfall enthält die Variable $link nun die bidirektionale Verbindung zur Datenbank. $link wird von nun an als Übergabeparameter aller php-Funktionen, die mit der Datenbank arbeiten, benötigt. Abb. 10.2 gibt übrigens einen Überblick über die MySql-Funktionen von php.

Als nächstes wird die Datenbank ausgewählt. Ein Datenbanksystem kann beliebig viele[7] Datenbanken enthalten. Abb. 10.3 zeigt die Datenbanken eines auf einem Server des eBusiness-Labors installierten Datenbank-Systems.

Die Auswahl erfolgt mittels des SQL-Kommandos `use`.

```
$query="use intranet";
mysql_query($query, $link);
```

In der ersten Zeile obigen php-Codes wird das SQL-Kommando auf der Variablen $query abgespeichert. Hier wird die Datenbank intranet ausgewählt. Dies ist übrigens die unserem Informationssystem zu Grundeliegende Datenbank. Mit dem Kommando `mysql_query` wird das SQL-Kommando dann an die Datenbank geschickt und dort ausgeführt. Wie wir sehen, erwartet mysql_query zwei Übergabeparameter:

- $query: Das SQL-Kommando, das vom Datenbank-System ausgeführt werden soll.

[5] Oder eben auch nicht, falls etwas schiefgegangen ist.

[6] SQL ist nicht Thema dieser Ausarbeitung. Unterlagen zu SQL finden Sie in den Datenbank-Unterlagen meines Kollegen Johannes, die über meine Internet-Seite verfügbar sind.

[7] Wird natürlich durch Plattenspeicher und Leistungsfähigkeit der Server-Hardware begrenzt.

XXXIII. MySQL functions

These functions allow you to access MySQL database servers.

More information about MySQL can be found at http://www.mysql.com/.

Table of Contents

mysql_affected_rows ? Get number of affected rows in previous MySQL operation
mysql_change_user ? Change logged in user on active connection
mysql_close ? close MySQL connection
mysql_connect ? Open a connection to a MySQL Server
mysql_create_db ? Create a MySQL database
mysql_data_seek ? Move internal result pointer
mysql_db_query ? Send an MySQL query to MySQL
mysql_drop_db ? Drop (delete) a MySQL database
mysql_errno ? Returns the number of the error message from previous MySQL operation
mysql_error ? Returns the text of the error message from previous MySQL operation
mysql_fetch_array ? Fetch a result row as an associative array
mysql_fetch_field ? Get column information from a result and return as an object
mysql_fetch_lengths ? Get the length of each output in a result
mysql_fetch_object ? Fetch a result row as an object
mysql_fetch_row ? Get a result row as an enumerated array
mysql_field_name ? Get the name of the specified field in a result
mysql_field_seek ? Set result pointer to a specified field offset
mysql_field_table ? Get name of the table the specified field is in
mysql_field_type ? Get the type of the specified field in a result
mysql_field_flags ? Get the flags associated with the specified field in a result
mysql_field_len ? Returns the length of the specified field
mysql_free_result ? Free result memory
mysql_insert_id ? Get the id generated from the previous INSERT operation
mysql_list_fields ? List MySQL result fields
mysql_list_dbs ? List databases available on on MySQL server
mysql_list_tables ? List tables in a MySQL database
mysql_num_fields ? Get number of fields in result
mysql_num_rows ? Get number of rows in result
mysql_pconnect ? Open a persistent connection to a MySQL Server
mysql_query ? Send an SQL query to MySQL
mysql_result ? Get result data
mysql_select_db ? Select a MySQL database
mysql_tablename ? Get table name of field

Abbildung 10.2: Die mysql-Funktionen von php

- $link: Die bidirektionale Verbindung zur Datenbank.

Nun ist die Datenbank ausgewählt. Wir können weitere, beliebige SQL-Befehle an die Datenbank schicken:

```
$query="select * from room";
$result=mysql_query($query, $link);
```

Obiges SQL-Kommando selektiert alle Datensätze der Tabelle room aus der Datenbank intranet. `mysql_query` schickt die Abfrage an die Datenbank. Das Ergebnis steht auf der Variablen $result. Beispiel 10.1 zeigt eine Ausgabe obigen SQL-Kommandos in einer direkten Schnittstelle des Datenbanksystems.

Beispiel 10.1 *Ausgabe eines SQL-Kommandos*

```
mysql> select * from room;
+---------+----------+----------------+
| room_nr | name     | group_of_seats |
+---------+----------+----------------+
|       1 | 01-36    |              2 |
|       2 | 01-37    |              2 |
|       3 | 0-38     |              5 |
|       4 | 0-39     |              5 |
|       5 | 1-39     |              3 |
```

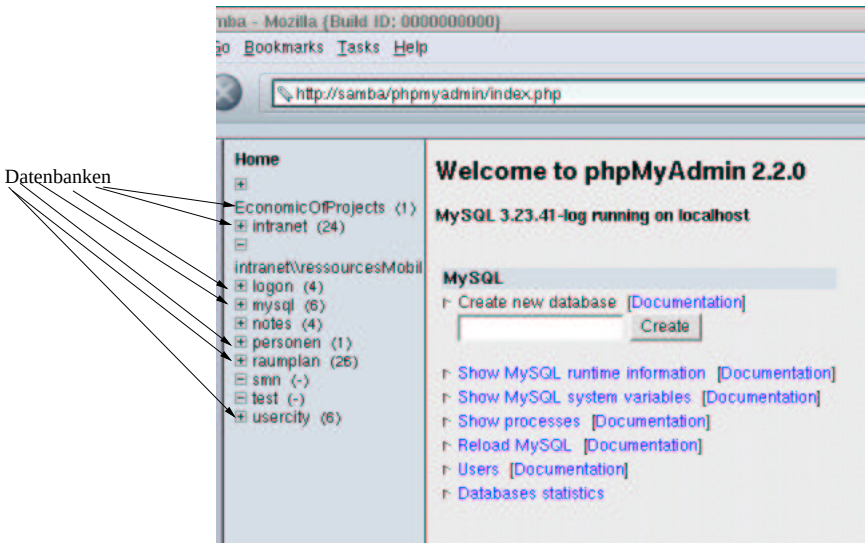


Abbildung 10.3: Datenbanken eines Datenbanksystems

```
|       6 | 1-40     |               2 |
|       7 | 1-41     |               5 |
|       8 | 2-33     |               3 |
|       9 | 2-34     |               2 |
|      10 | 2-35     |               5 |
|      11 | 3-33     |               3 |
|      12 | 3-34     |               2 |
|      13 | 3-35     |               5 |
|      14 | 4-23     |               3 |
|      15 | 4-24     |               2 |
|      16 | 4-25     |               5 |
|      17 | 5-24     |               1 |
|      18 | 5-26     |               5 |
|      19 | HS 6     |               0 |
|      20 | HS 7     |               0 |
|      21 | SR A1-26 |               0 |
|      22 | NN       |               0 |
|      23 | C4-07    |               0 |
|      24 | C3-12    |               0 |
|      25 | C3-13    |               0 |
|      26 | C2-15    |               0 |
|      27 | 0-36     |               1 |
+---------+----------+-----------------+
27 rows in set (0.00 sec)
```

Die Ergebnismenge ist also eine Tabelle, in der die durch die Abfrage ermittelten Datensätze untereinander aufgeführt sind. Mit der Funktion `mysql_result` können wir einzelne Elemente der Ergebnismenge ansprechen.

```
$ersterDatensatzErstesFeld=mysql_result($result,0,0);
```

Die Funktion `mysql_result` erwartet also als Ersten Übergabeparameter eine von `mysql_query`

erzeugte Ergebnismenge. Die nächsten beiden Übergabeparameter sind Zeile und Spalte des gewünschten Wertes der Ergebnismenge, wobei, wie häufig in der Informatik, von Null hochgezählt wird. $ersterDatensatzErstesFeld hat in unserem Beispiel den Wert 1 (vgl. Beispiel 10.1). Um den Wert “1-39” zu erhalten, müsste das Kommando

```
$ersterDatensatzErstesFeld=mysql_result($result,4,1);
```

abgesetzt werden (vgl. Beispiel 10.1).

10.3 Erste Lösung, Stringbehandlung

Die Datenbank

Zunächst müssen wir das Datenbankmodell entwickeln. Das machen wir mit Entity Relationship-Modellierung (ERM)[8]. Hier ist dies ganz einfach: Es gibt nur eine Tabelle, die nennen wir “waehrung”. Die Attribute sind: “waehrung_nr”, “name”, “kurs”. Abb. 10.4 zeigt das ERM.

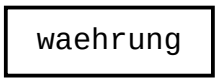


Abbildung 10.4: Das ERM zur Euro-Dollar-Problemstellung

Zunächst müssen wir eine Datenbank und dann die Tabelle im Datenbanksystem anlegen. Dies können wir entweder direkt oder mit phpmyadmin tun.

Das Ergebnis ist in Abb. 10.5 zu sehen.

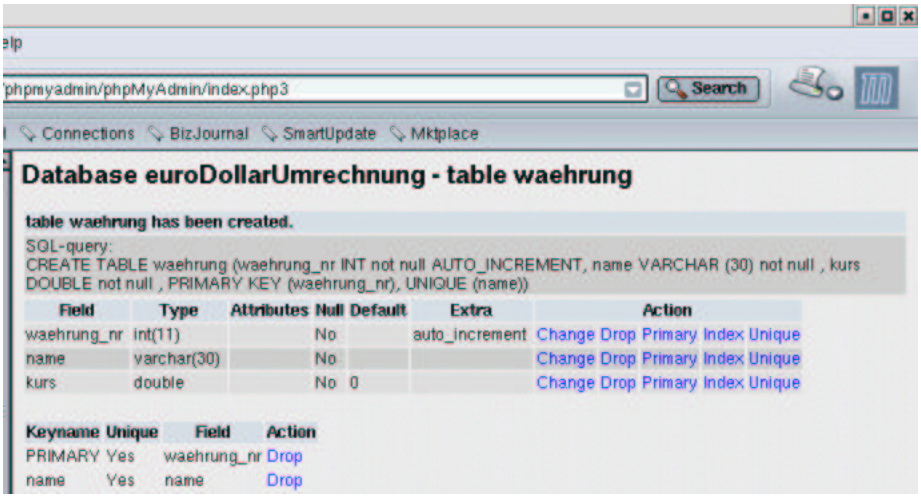


Abbildung 10.5: Die Tabelle Währung

“waehrung_nr” ist der selbst hoch zählende Primärschlüssel. “name” ist auf unique gesetzt, damit ist garantiert, dass der Name einer Währung innerhalb der Tabelle eindeutig ist. Zum Abschluss unserer Datenbankaktivitäten tragen wir einen Datensatz in die Tabelle ein. Dies zeigt Abb. 10.6.

Beachten Sie, dass Sie für “waehrung_nr” keinen Wert angeben müssen. Weil “waehrung_nr” auf “autoincrement” steht, wird der Wert von “waehrung_nr” von der Datenbank vergeben.

[8] Ebenfalls in den Unterlagen von Herrn Johannes zu finden.

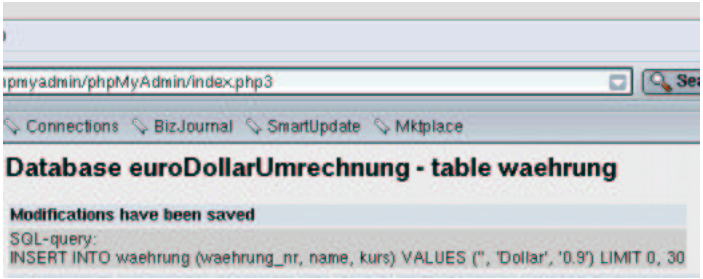


Abbildung 10.6: Der Eintrag eines Datensatzes

Die Seite zur Kurspflege

Nun folgt die Entwicklung der Seite zur Pflege des Dollarkurses. Funktion und Aussehen dieser Seite könnten wir, wie in Abb. 10.7 dargestellt[9], programmieren.

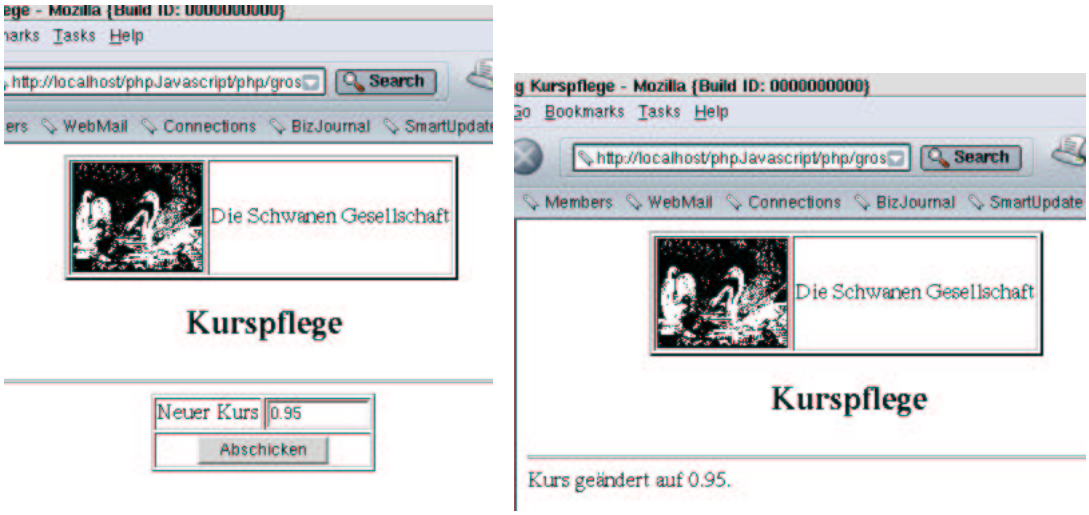


Abbildung 10.7: Die Seite zur Kurspflege

Das Programm dazu sieht im ersten Ansatz so aus:

Beispiel 10.2 *Die Kurspflege Anwendung*

```
<!-- Programm zur Euro-Dollar Umrechnung
     Teil Das grosse Ganze
     Kurspflege
  Dateiname: grosseGanze1/kurspflege.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
</head>
<body>
```

[9] Ich bin halt kein Designer!

```
    <table border=2 align="center">
             <tr>
               <td>
                    <img src="images/swan.jpg" width=96 height=80
                                                align="left" border=1>
               </td>
               <td>
                    Die Schwanen Gesellschaft
               </td>
             </tr>
    </table>
    <h2 align="center">
        Kurspflege
    </h2>
    <hr>
<?php
    // Wir pruefen ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='kurspflege' action='$PHP_SELF' method='post'>";
?>
            <table border align="center">
                <tr>
                    <td>
                        Neuer Kurs
                    </td>
                    <td>
                        <input type="text" name="kurs" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        // Verbindung zur Datenbank herstellen
        $link=mysql_pconnect("localhost", "root", "");
        // Datenbank euroDollarUmstellung einstellen
        $query="use euroDollarUmrechnung";
        mysql_query($query, $link);
        // update sql-Kommando
        $updateQuery ="update waehrung set kurs='$kurs' ";
        $updateQuery.="where name='Dollar'";
        mysql_query($updateQuery, $link);
        echo "Kurs ge&auml;ndert auf $kurs.";
    }
?>
```

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

Neues beginnt hier im `else`-Teil. In der Zeile

```
$link=mysql_pconnect("localhost", "root", "");
```

wird eine Verbindung zum Datenbank-System aufgebaut. Der Server, auf dem das Datenbank-System implementiert ist, ist derjenige, auf dem auch das php-Script läuft (localhost). Der Benutzer, der angemeldet wird, heißt root. Er hat kein Passwort (“”). Die Kennung der Verbindung ist auf der Variablen $link abgespeichert.

```
$query="use euroDollarUmrechnung";
mysql_query($query, $link);
```

In der ersten obigen Zeile wird das SQL-Kommando zum Anmelden der Datenbank euroDollarUmrechnung der Variablen $query zugewiesen. In der zweiten Zeile wird das Kommando an die Datenbank geschickt und dort ausgeführt.

```
$updateQuery ="update waehrung set kurs='$kurs' ";
$updateQuery.="where name='Dollar'";
mysql_query($updateQuery, $link);
```

In den ersten beiden oben dargestellten Zeilen wird das SQL-Kommando zum Update der Datenbank (Änderung des Dollar-Kurses) zusammengestellt. Beachten Sie die Konstruktion:

```
$updateQuery.="where name='Dollar'";
```

Dies ist einfach eine abkürzende Schreibweise für die Stringverkettung (.=). Dies hatten wir in Kapitel 4 besprochen. Variablen oder Zeichenketten, die einem Feld der Datenbank zugewiesen werden sollen, werden in Apostrophe (”) eingeschlossen.

Zwei Schönheitsfehler hat die Anwendung zur Kurspflege noch:

- Die Benutzer können keine Kommas als Dezimaltrenner eingeben.
- Es wird nicht überprüft, ob der eingegebene Wert eine Zahl ist.

Eingabe-Überprüfungen realisieren wir, wie in Kapitel 9 dargestellt, in JavaScript. Dadurch ändert sich auch die Definition des Formulars in der php-Datei (vgl. Beispiel 9.16), da die Übertragung des Formulars ja jetzt durch JavaScript geschehen muss. Die Funktion zur Überprüfung, ob eine Eingabe eine Zahl ist, hatten wir bereits implementiert. Sie ist in Beispiel 9.14 dargestellt.

Nun müssen wir noch mit der Eingabe eines Kommas als Dezimaltrenner klarkommen. Aber auch das ist gar nicht so schwer. Wir untersuchen die Eingabe des Benutzers auf die Eingabe eines Kommas. Finden wir ein Komma, ersetzen wir es durch einen Punkt. Die Realisierung findet sich in Beispiel 10.3.

Beispiel 10.3 *Kommas in Punkte umwandeln*

```
// Funktionen zur Stringbearbeitung
//
// Dateiname: stringbearbeitung.js
function komma2punkt(str)
```

```
{
    str=str.replace(/,/g,".");
    return str;
}
```

Strings in JavaScript sind hybride Variablen. Das bedeutet:

- Benutze ich einen String ganz normal als Variable, wird er von JavaScript auch so behandelt. Ich kann ihn mit anderen Strings verketten, einem anderen String zuweisen, vergleichen und was der Dinge mehr sind, die man mit Variablen machen kann und die Sie bereits alle gelernt haben.
- Schließe ich aber an eine String-Variable einen Punkt an, denkt JavaScript, ich möchte den String gerne als Objekt behandeln[10]. JavaScript konvertiert den String dann in ein String-Objekt und ich kann alle Methoden der String-Klasse auf meine String-Variable anwenden[11]. Benutze ich das String-Objekt, zu dem meine Variable ja geworden ist, wieder als normale Variable, macht JavaScript alles rückgängig und ich habe wieder meine normale Variable.

So erklärt sich also, dass ich in der Zeile

```
str=str.replace(/,/g,".");
```

einen Methodennamen über einen Punkt mit einer String-Variablen verbinden kann. `replace` ist nun eine Methode der String-Klasse und ersetzt in dem String-Objekt, das diese Methode aufruft, den ersten Übergabeparameter durch den zweiten. Ein wenig gewöhnungsbedürftig ist die Syntax, wie die Übergabeparameter angegeben werden müssen. Der erste Übergabeparameter (die Zeichenkette, die ersetzt werden soll) wird in Schrägstriche (Slash, /) eingeschlossen. Steht hinter dem schließenden Schrägstrich ein g, so bedeutet dies, dass die zu ersetzende Zeichenkette jedesmal, wenn sie auftaucht, ersetzt wird. Fehlt das g hinter dem schließenden Schrägstrich, wird die Zeichenkette nur einmal ersetzt. Die Zeichenkette, die die erste ersetzen soll, wird, wie wir das gewohnt sind, in Anführungszeichen (“”) eingeschlossen[12]. Was die in Beispiel 10.3 dargestellte Funktion komma2punkt() also tut, ist, jedes Komma im ihr übergebenen String in einen Punkt umzuwandeln. Die Funktion komma2punkt() speichern wir im Verzeichnis javascript unterhalb des Verzeichnisses, in dem unsere kurspflege.php liegt.

Jetzt müssen wir also nur noch das JavaScript-Script schreiben, das unsere Funktionen komma2punkt() und istKeineZahl() aufruft.

Beispiel 10.4 *Die Kurspflege Anwendung verbessert*

```
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
```

[10] Was ich dann auch besser möchte!

[11] Die man wiederum in der JavaScript-Referenz nachschlagen kann, die String-Methoden!

[12] Für diese merkwürdig erscheinende Syntax gibt es tatsächlich einen Grund. Ich habe hier nämlich nicht die ganze Wahrheit verraten. Tatsächlich erzeugen die Schrägstriche einen regulären Ausdruck (regular Expression, Reg Exp). Reguläre Ausdrücke kommen aus dem Unix und sind eine mächtige Waffe bei der Stringbearbeitung. Mit ihnen kann man nicht nur einfache Ersetzungen durchführen, sondern auch auf “recht einfache Art” (wenn man es denn kann), Bedingungen formulieren wie: Ersetze abc durch xyz, und zwar nur dann, wenn in dem Wort, was abc enthält, nur die Buchstaben abcde vorkommen und wenn das Wort das dritte im Absatz ist und nicht mehr als 8 Buchstaben enthält und jedes dritte Mal lass aus, selbst wenn alle anderen Bedingungen zutreffen ... Reguläre Ausdrücke werden wir aber nicht weiter behandeln.

```
    Kurspflege
  Dateiname: grosseGanze1/kurspflege2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript"
            src="./javascript/stringbearbeitung.js">
  </script>

  <script language = "JavaScript">
    function teste()
    {
        document.kurspflege.kurs.value=komma2punkt(
                                              document.kurspflege.kurs.value);

        if(istKeineZahl(document.kurspflege.kurs,
                        "Kurs ist keine Zahl"))
        {
            return false;
        }
        document.kurspflege.method="post";
        document.kurspflege.action=document.kurspflege.action.value;
        document.kurspflege.submit();
    }
  </script>
</head>
<body>
    <table border=2 align="center">
             <tr>
               <td>
                    <img src="images/swan.jpg" width=96 height=80
                                              align="left" border=1>
               </td>
               <td>
                    Die Schwanen Gesellschaft
               </td>
             </tr>
    </table>
    <h2 align="center">
        Kurspflege
    </h2>
    <hr>
<?php
    // Wir pruefen ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
?>
        <form name='kurspflege'>
            <input type="hidden" name="action"
```

```
                    value="<?php echo $PHP_SELF ?>" >
            <table border align="center">
                <tr>
                    <td>
                        Neuer Kurs
                    </td>
                    <td>
                        <input type="text" name="kurs" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="button" name="Button1" onClick="teste()"
                                                            value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        // Verbindung zur Datenbank herstellen
        $link=mysql_pconnect("localhost", "root", "");
        // Datenbank euroDollarUmstellung einstellen
        $query="use euroDollarUmrechnung";
        mysql_query($query, $link);
        // update sql-Kommando
        $updateQuery ="update waehrung set kurs='$kurs' ";
        $updateQuery.="where name='Dollar'";
        mysql_query($updateQuery, $link);
        echo "Kurs ge&auml;ndert auf $kurs.";
    }
?>
</body>
</html>
```

Die Zeilen

```
        <form name='kurspflege'>
            <input type="hidden" name="action"
                        value="<?php echo $PHP_SELF ?>" >
```

zusammen mit der Umimplementierung des Button

```
    <input type="button" name="Button1" onClick="teste()"
                                        value="Abschicken">
```

sorgen dafür, dass unser Formular nur über JavaScript abgeschickt werden kann.

```
      onClick="teste()"
```

sorgt dafür, dass die JavaScript-Funktion teste() aufgerufen wird, wenn der Benutzer auf den Button clickt. Die Funktion teste() ist im <head>-Teil der php-Datei deklariert. In teste() wird zunächst die Funktion komma2punkt auf den Wert des Input-Feldes kurs angewendet. Das bedeutet, etwaige Kommas, die unsere Benutzer eingegeben haben werden durch Punkte ersetzt. Das Ergebnis von komma2punkt wird in das Input-Feld zurückgeschrieben:

```
document.kurspflege.kurs.value=komma2punkt(
                              document.kurspflege.kurs.value);
```

Danach checkt unsere bereits in Kapitel 9 entwickelte Funktion istKeineZahl(), ob die Eingabe des Benutzers eine Zahl war. Beachten Sie die Reihenfolge. Wir müssen zuerst die Kommas durch Punkte ersetzen, dann erst können wir überprüfen, ob die Eingabe eine Zahl war. Der Grund ist einfach: istKeineZahl() benutzt die JavaScript interne Methode isNaN(). Und isNaN würde bei Kommas in Zahlen nie Zahlen erkennen. War alles okay, setzt teste() die Anfragemethode (post), setzt die "action" auf die php-Datei selbst und schickt das Formular ab. Dies entspricht aber Beispiel 9.16 aus Kapitel 9.

Die Produktseite

Wir starten mit einem grafischen Prototyp der Seite. Unsere Benutzer können dann beurteilen, ob das Design und die Funktionalität der Seite das ist, was sie erwarten. Die erste Näherung der Produktseite zeigt Abb. 10.8.

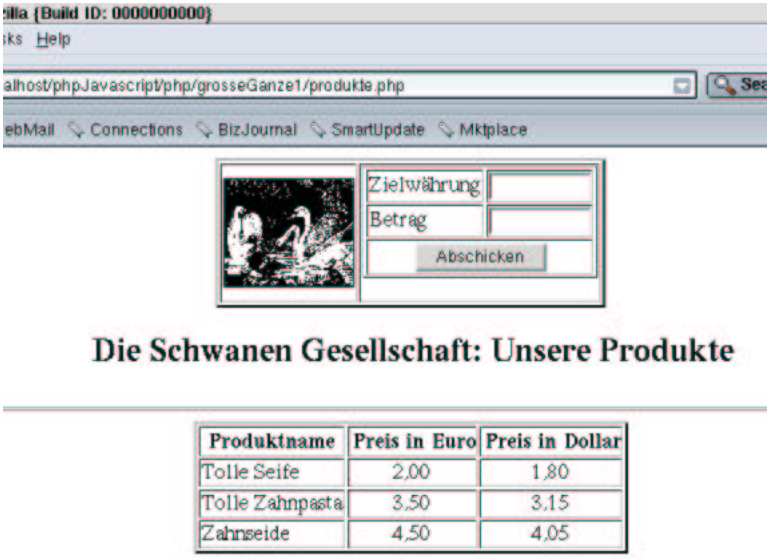


Abbildung 10.8: Produktseite, die erste Näherung

Allein, der Prototyp findet keine Gnade vor den Augen unserer Benutzer (der Marketingabteilung). Die meinen nämlich (nicht ganz zu unrecht), dass der Euro-Dollar-Umrechner dazu dienen soll, Surfer auf unsere Produktseite zu ziehen, damit die, während sie den Rechner benutzen, unsere Produkte im Auge haben und dann vielleicht die "tolle Seife" bestellen. Und bei unserem Prototyp sieht es ja so aus, dass sich für die Nutzer, wenn sie auf "Abschicken" clicken, eine neue Seite öffnet. Außerdem findet Die Abteilung Marketing die Bedienung zu kompliziert. Besser ist, wenn das eine Input-Feld mit Euro beschriftet ist und das andere mit Dollar. Nimmt der Benutzer einen Eintrag oder

eine Änderung vor, wird gerechnet und das Ergebnis im jeweils anderen Feld dargestellt. Es ergibt sich das Design aus Abb. 10.9.

Abbildung 10.9: Produktseite, so soll es aussehen

Uns stellt das natürlich vor diverse Probleme. Fangen wir mit den einfachen an. Zunächst sollen die aus den Europreisen berechneten Dollarpreise auf der Produktseite unserem Anforderungskatalog entsprechend mit zwei Nachkommastellen und dem Komma als Dezimaltrenner ausgegeben werden. Das ist in php nicht wirklich schwierig. In php gibt es diverse Funktionen für formatierte Ausgaben. Für unsere Zwecke geeignet ist number_format(). number_format() erwartet vier Übergabeparameter:

- Der erste Übergabeparameter ist die Zahl, die formatiert ausgegeben werden soll.
- Der zweite Übergabeparameter gibt die Anzahl Nachkommastellen an.
- Der dritte Übergabeparameter repräsentiert den Dezimaltrenner.
- Der vierte Übergabeparameter ist der Tausendertrenner.

Als nächstes müssen wir den Dollarkurs aus der Datenbank lesen. Das können wir aber auch. Wir sprechen mit mysql_pconnect das Datenbank-System an, wählen mit “use euroDollarUmrechnung” eben jene Datenbank aus, schicken einen select-Befehl an die Datenbank, und holen aus der Ergebnismenge das gewünschte Feld.

Ansonsten können wir für den Produktbereich der gewünschten php-Seite die Lösung aus Beispiel 7.10 fast unverändert übernehmen.

Etwas anders ist die Situation beim Euro-Dollar-Umrechner. Die Logik können wir mit JavaScript eigentlich leicht lösen. Das Formular entspricht nämlich Beispiel 9.13. Es gibt nur ein Problem: In Beispiel 9.13 hatten wir den Dollarkurs fest in das JavaScript-Programm aufgenommen. Jetzt steht er in einem Feld einer Tabelle einer Datenbank auf einem unserer Server. Und JavaScript läuft im Browser des Clients. Es gibt keinerlei Möglichkeit, von JavaScript aus auf Datenbanken des Servers zuzugreifen. Und hier machen wir den ganz großen Trick: Wir lesen beim Aufbau der Produktseite den Dollarkurs mit php aus der Datenbank. Dann schreiben wir mit dem php-echo-Befehl das ganze

JavaScript-Script in die Datei. Das hört sich jetzt ungeheuer kompliziert an, ist es aber gar nicht so wirklich. Wenn gleich der Code im Einzelnen besprochen wird, werden Sie das (wie ich hoffe) leicht verstehen. Und zum Schluss müssen die Ausgaben noch mit zwei Dezimalstellen und dem Komma als Dezimaltrenner in die Textfelder von Abb. 10.9 geschrieben werden. In JavaScript gibt es keine Funktionen zur formatierten Ausgabe von Zahlen. Also müssen wir selber eine schreiben, die das macht.

Damit beginnen wir. Wir erweitern die JavaScript-Datei “stringbearbeitung.js” um eine Funktion zur schönen Ausgabe von Zahlen. Es handelt sich dabei um die in Beispiel 10.5 dargestellte Funktion “schoeneAusgabe()”.

Beispiel 10.5 *Komma als Dezimaltrenner in Javascript*

```
// Funktionen zur Stringbearbeitung
//
// Dateiname: stringbearbeitung2.js
function komma2punkt(str)
{
    str=str.replace(/,/g,".");
    return str;
}
function schoeneAusgabe(str)
{
    b=str.indexOf (".");
    if (b==-1)
    {
        str=str+ ",00"; // keine Nachkommastellen
    }
    else
    {
        if((str.substring(b+1,str.length)).length == 1)
        {
            str=str.substring(0,b)+","+str.substring(b+1,b+2)+"0";
        }
        else
        {
            str=str.substring(0,b)+","+str.substring(b+1,b+3);
        }

    }
    return str;
}
```

“schoeneAusgabe()” erwartet einen Übergabeparameter: Den String, der einen Punkt enthalten kann und der demzufolge umgewandelt werden muss.

```
function schoeneAusgabe(str)
```

Bei der Umwandlung wenden wir weitere Methoden der Stringklasse an:

```
b=str.indexOf (".");
```

gibt als Ergebnis die Stelle zurück, an der der erste Punkt im String str steht. Enthält der String keinen Punkt, wird -1 zurückgegeben. In diesem Fall gibt es keine Nachkommastellen und ,00 wird angehängt.

Gibt es hingegen einen Punkt, teilen wir den String mit der String-Methode substring auf. substring extrahiert einen Teilstring aus dem String-Objekt. Der Teilstring beinhaltet das Zeichen am ersten substring übergebenen Parameter, sowie alle Zeichen zwischen dem ersten und dem zweiten Parameter, das Zeichen am zweiten übergebenen Parameter allerdings nicht.

```
str.substring(0,b)
```

enthält also alle Zeichen bis zum Punkt, den Punkt selber aber nicht mehr. Hierbei müssen wir noch unterscheiden, ob es eine oder mehrere Nachkommastellen gibt. Bei einer Nachkommastelle müssen wir nämlich eine Null anfügen. Sind es zwei oder mehr, werden die über zwei hinausgehenden Stellen abgeschnitten. Um dies zu ermitteln, benutzen wir die Eigenschaft length des String-Objekts. length enthält die Anzahl Zeichen eines Strings.

```
str.substring(b+1,str.length)).length
```

ist also folgendermaßen zu lesen: Bilde einen Substring des übergebenen Strings, der direkt nach dem Punkt beginnt (Stelle b+1) und alle Zeichen bis zum letzten Zeichen enthält (str.length) und berechne von dem so erzeugten String die Länge. Ist diese Eins, so setzen wir den String wieder zusammen, mit der Ausnahme, dass wir an die Stelle wo der Punkt stand, ein Komma einsetzen. Zusätzlich fügen wir eine Null an:

```
str=str.substring(0,b)+","+str.substring(b+1,b+2)+"0";
```

Gibt es zwei oder mehr Stellen nach dem Komma, fügen wir das Komma ein und schneiden nach der zweiten Stelle ab:

```
str=str.substring(0,b)+","+str.substring(b+1,b+3);
```

Beachten Sie, dass diese Funktion nicht ganz korrekt ist, da sie nach der zweiten Stelle abschneidet. Eigentlich müsste dort gerundet werden. Sie können sich überlegen, wie man vorgehen müsste, um die Funktion vollständig richtig zu implementieren!

Als nächstes betrachten wir die Funktionen zur Umrechnung von Euro in Dollar bzw. umgekehrt. Dabei können wir auf die bereits in Beispiel 7.7 erstellte Umrechnungsfunktion zurückgreifen. Wir müssen sie nur an die neuen Gegebenheiten anpassen:

Beispiel 10.6 *Euro-Dollar-Umrechnung mit Datenbank*

```
<?php
// Funktion zur Dollar-Euro oder Euro-Dollar Umrechnung
// Datei:euroDollarUmrechnung.inc.php
// Verzeichnis: includes
    function holeDollarKurs()
    {
        $link=mysql_pconnect("localhost", "root", "");
        $query="use euroDollarUmrechnung";
        mysql_query($query, $link);
        $query="Select kurs from waehrung where name='Dollar'";
        $result=mysql_query($query, $link);
        $kurs=mysql_result($result,0,0);
        return $kurs;
```

```
    }
    function euroDollarUmrechnung($zielwaehrung, $betrag)
    {
        $kurs=holeDollarKurs();
        if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
        {
            $dollarbetrag=$kurs*$betrag;
            return $dollarbetrag;
        }
        if(($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
        {
            $eurobetrag=(1/$kurs)*$betrag;
            return  $eurobetrag;
        }
    }
    function erzeugeEuroDollarUmrechnungsScript()
    {
        $kurs=holeDollarKurs();
        echo "<script language=\"JavaScript\">\n";
            echo "function euroDollarUmrechnung(zielwaehrung, betrag)\n";
            echo "{\n";
                echo "var kurs=$kurs;\n";
                echo "if((zielwaehrung==\"Dollar\")||(zielwaehrung==\"dollar\"))\n";
                echo "{\n";
                    echo "dollarbetrag=kurs*betrag;\n";
                    echo "return(dollarbetrag);\n";
                echo "}\n";
                echo "if((zielwaehrung==\"Euro\")||(zielwaehrung==\"euro\"))\n";
                echo "{\n";
                    echo "eurobetrag=(1/kurs)*betrag;\n";
                    echo "return(eurobetrag);\n";
                echo "}\n";
            echo "}\n";
        echo "</script>\n";
    }
?>
```

```
    $kurs=holeDollarKurs();
```

ist die einzige Zeile, in der sich Beispiel 10.6 von Beispiel 7.7 unterscheidet. Das ist eigentlich auch einsichtig, denn, wenn wir den Dollarkurs aus der Datenbank gelesen haben, ist der Algorithmus ja auch identisch. Betrachten wir nun holeDollarKurs(): Die ersten Zeilen kennen wir schon. Sie stellen die Verbindung zum Datenbank-System her und stellen die Datenbank euroDollarUmrechnung ein.

```
    $query="Select kurs from waehrung where name='Dollar'";
```

ist das SQL-Kommando, um den Dollarkurs aus der Datenbank zu lesen. Wie bereits weiter oben besprochen, erhalten wir von der Datenbank eine Ergebnismenge zurück, die wir mit mysql_result bearbeiten können. Da die Ergebnismenge nur ein Element enthalten kann, gibt es in der Ergebnismenge nur eine Zeile und eine Spalte. Daher erhalten wir mit

```
    $kurs=mysql_result($result,0,0);
```

den Dollarkurs.

Ganz neu ist erzeugeEuroDollarUmrechnungsScript(). Denken wir uns bei dieser Funktion einmal die echo-Befehle am Anfang jeder Zeile weg, so sehen wir, dass es zu Beispiel 7.12 eigentlich nur einen Unterschied gibt: Die JavaScript-Variable kurs wird in Beispiel 10.6 mit dem Inhalt der php-Variablen $kurs besetzt, auf der vorher der aus der Datenbank gelesene Dollarkurs gespeichert wurde. Die anderen Änderungen sind marginal: Der \ vor den Anführungszeichen sorgt dafür, dass die Anführungszeichen nicht als Ende des jeweiligen Strings interpretiert werden[13]. Das am Ende einer Zeile eingefügte \n ist das Zeichen für den Zeilenvorschub. Dies sorgt nur dafür, dass die "view source"-Funktion des Browsers den JavaScript-Source-Code untereinander und nicht in einer Zeile[14] anzeigt. Wenn also erzeugeEuroDollarUmrechnungsScript() in einer html-Datei aufgerufen wird, wird, wie Sie ja wissen, der php-Code durch seinen Output ersetzt und der Output ist in diesem Fall eine JavaScript-Funktion, die Euros in Dollar und umgekehrt umrechnen kann.

Jetzt sind wir soweit, uns die Implementierung der Produktseite ansehen zu können. Beispiel 10.7 zeigt sie.

Beispiel 10.7 *Die Produktseite mit Euro-Dollar-Umrechnung*

```
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Produktseite mit Euro-Rechner
  Dateiname: grosseGanze1/produkte.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
<?php
    require_once("./includes/euroDollarUmrechnung.inc.php");
    erzeugeEuroDollarUmrechnungsScript();
?>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript"
            src="./javascript/stringbearbeitung2.js">
  </script>
  <script language = "JavaScript">
    function dollar2euro()
    {
            document.euro6.dollar.value=komma2punkt
                            (document.euro6.dollar.value);
            if(istKeineZahl(document.euro6.dollar,
                                "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            {
                document.euro6.euro.value=
                        schoeneAusgabe(euroDollarUmrechnung(
```

[13] Sie qerden durch den \ escaped, wir hatten das ja viel früher schon einmal besprochen.

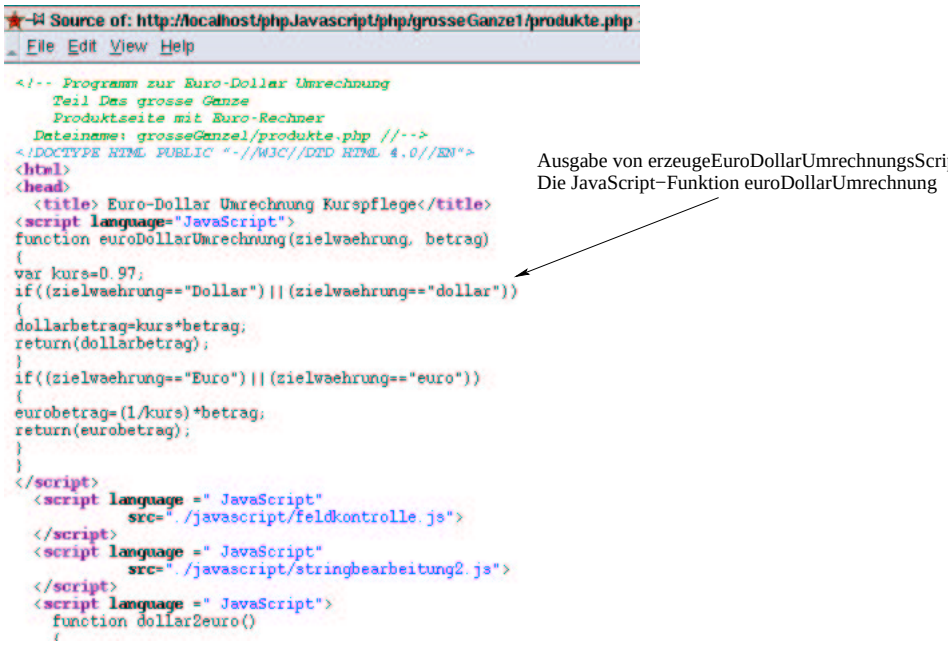
[14] Ich werde Ihnen nämlich noch einen Screenshot zeigen, in dem man sehen kann, dass tatsächlich eine JavaScript-Funktion an den Browser übertragen wird.

```
                            "euro", document.euro6.dollar.value).toString());
            }
    }
    function euro2dollar()
    {
            document.euro6.euro.value=komma2punkt
                            (document.euro6.euro.value);
            if(istKeineZahl(document.euro6.euro,
                                "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            {
                document.euro6.dollar.value=
                        schoeneAusgabe(euroDollarUmrechnung(
                            "dollar", document.euro6.euro.value).toString);
            }
    }
    </script>
</head>
<body>
    <table border=2 align="center">
              <tr>
                <td>
                     <img src="images/swan.jpg" width=96 height=80
                                                 align="left" border=1>
                </td>
                <td valign="bottom">
                     <form name='euro6'>
                         <table border>
                             <tr>
                                 <td>
                                     Dollar
                                 </td>
                                 <td>
                                     <input type="text" name="dollar"
                                         onChange="dollar2euro()" size=12>
                                 </td>
                             </tr>
                             <tr>
                                 <td>
                                     Euro
                                 </td>
                                 <td>
                                     <input type="text" name="euro"
                                         onChange="euro2dollar()" size=12>
                                 </td>
                             </tr>
                         </table>
                     </form>
                   </td>
              </tr>
    </table>
```

```
    <h2 align="center">
        Die Schwanen Gesellschaft: Unsere Produkte
    </h2>
    <hr>
    <table border="2" align="center">
    <tr>
        <th> Produktname </th>
        <th> Preis in Euro </th>
        <th> Preis in Dollar </th>
    </tr>
    <tr>
        <td> Tolle Seife </td>
        <td align="center"> 2,00 </td>
        <td align="center">
    <?php
            echo(number_format(euroDollarUmrechnung("Dollar", 2),2,","," "))
    ?>
        </td>
    </tr>
    <tr>
        <td> Tolle Zahnpasta </td>
        <td align="center"> 3,50 </td>
        <td align="center">
    <?php
            echo(number_format(euroDollarUmrechnung("Dollar", 3.5),2,","," "))
    ?>
        </td>
    </tr>
    <tr>
        <td> Zahnseide </td>
        <td align="center"> 4,50</td>
        <td align="center">
    <?php
            echo(number_format(euroDollarUmrechnung("Dollar", 4.5),2,","," "));
    ?>
        </td>
    </tr>
    </table>
</body>
</html>
```

Direkt im <head>-Teil wird erzeugeEuroDollarUmrechnungsScript() aufgerufen. Dies bedeutet, erzeugeEuroDollarUmrechnungsScript() wird durch den erzeugten JavaScript-Code ersetzt. Der Browser erhält die in Abb. 10.10 dargestellten Daten.

Der Rest der Produktseite besteht aus der Zusammensetzung der Beispiele 7.10 und 9.11. Der Unterschied zu Beispiel 9.11 besteht darin, dass von den mit den onChange-Event-Handlern der Text-Felder verbundenen Funktionen zunächst komma2punkt() aufgerufen wird, da laut Anforderungskatalog Kommas als Dezimaltrenner zugelassen sind. Vor dem Zurückschreiben in das Input-Text-Feld wird ebenfalls im Unterschied zu Beispiel 9.11 schoeneAusgabe aufgerufen, damit die Ausgabe des errechneten Wertes mit zwei Dezimalstellen und dem Komma als Dezimaltrenner erfolgt. Betrachten wir diese Zeile genauer:




```
<!-- Programm zur Euro-Dollar Umrechnung
     Teil Das grosse Ganze
     Produktseite mit Euro-Rechner
  Dateiname: grosseGanze1/produkte.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
<script language="JavaScript">
function euroDollarUmrechnung(zielwaehrung, betrag)
{
var kurs=0.97;
if((zielwaehrung=="Dollar")||(zielwaehrung=="dollar"))
{
dollarbetrag=kurs*betrag;
return(dollarbetrag);
}
if((zielwaehrung=="Euro")||(zielwaehrung=="euro"))
{
eurobetrag=(1/kurs)*betrag;
return(eurobetrag);
}
}
</script>
  <script language =" JavaScript"
          src=" /javascript/feldkontrolle.js">
  </script>
  <script language =" JavaScript"
          src=" /javascript/stringbearbeitung2.js">
  </script>
  <script language =" JavaScript">
    function dollar2euro()
```

Abbildung 10.10: Die erzeugte JavaScript-Funktion

```
document.euro6.dollar.value=
        schoeneAusgabe(euroDollarUmrechnung(
                "dollar", document.euro6.euro.value).toString);
```

Zunächst erkennen wir wieder die Schachtelbarkeit von Funktionsaufrufen. Zunächst wird euroDollarUmrechnung() aufgerufen. schoeneAusgabe() verarbeitet das Ergebnis von euroDollarUmrechnung(). Ein bisschen stört noch ".toString". euroDollarUmrechnung() gibt aber eine Zahl zurück, schoeneAusgabe() hingegegen erwartet einen String. "toString" an eine eine Zahl enthaltende Variable sorgt dafür, dass der Inhalt der Variablen in einen String umformatiert wird. Warum JavaScript das in diesem Fall allerdings nicht automatisch macht, wo es doch sonst so viel automatisch macht, ist mir allerdings auch unklar.

Damit müsste der obere Teil, das Euro-Dollar-Umrechnungsformular, jetzt verständlich sein. Den zweiten Teil, die Umrechnung der Produktpreise, müssten Sie eigentlich direkt verstehen. Der Unterschied zu Beispiel 7.10 liegt nur darin, dass zwischen das echo-Kommando und den Aufruf von euroDollarUmrechnug() (diesmal die php-Funktion wohlgemerkt) ein Aufruf der Funktion number_format() geschaltet wurde. Der erste Parameter dieser Funktion ist das Ergebnis von euroDollarUmrechnug(), also die Zahl, die ausgegeben werden soll, der zweite die Anzahl Dezimalstellen (2), der dritte der Dezimaltrenner (,) und der vierte der Tausendertrenner (" ").

Zusammenfassung

Eine solche vergleichsweise doch überschaubare Aufgabenstellung erfordert bereits einen nicht unerheblichen Programmieraufwand. Mit Software wie MS-Frontpage kann man bei diesen Aufgabenstellungen "keinen Blumentopf mehr gewinnen". Die Aufgabe war eigentlich nur im Zusammenspiel von JavaScript, html, php und Datenbanktechnologie lösbar. Ganz abgesehen von der Anforderungsanalyse, die wir hier nur kurz behandelt haben. Die Gesamtlösung ist auf mehrere Dateien verteilt. Abb. 10.11 zeigt die Dateistruktur der Anwendung.

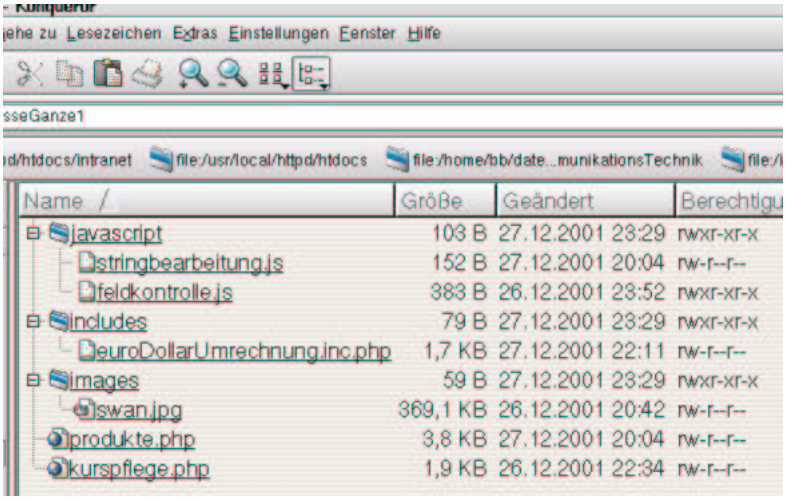


Abbildung 10.11: Die Dateistruktur der Anwendung

10.4 Verbesserte Lösung: Sessions und ein wenig Anwendungsarchitektur

Kaum ist das Produkt fertig und online gegangen, häufen sich die Probleme:

1. Die Marketing-Leute haben mitbekommen, dass es eine Seite gibt, über die man den Euro-Kurs ändern kann. Und machen das manchmal. Was natürlich die Mitarbeiter der Produktpflege nicht glücklich macht.

2. Sogar Kunden ändern manchmal den Euro-Kurs.

3. Das Software-Qualitätsmanagement fragt an, wie wir uns eine etwaige Umstellung auf ein anderes Datenbank-System vorstellen. Schließlich ist nichts für die Ewigkeit (schon gar nicht in deutschen Unternehmen). Bei unseren jetzigen Seiten müssten wir bei einer Änderung des Datenbank-Systems alle Seiten kontrollieren und ggf. anpassen.

4. Überdies findet unsere Anwendungsarchitektur keine Gnade beim Software-Qualitätsmanagement. Wir haben zwar eine Datei, in der wir Währungsfunktionen sammeln, nämlich euroDollarUmrechnung.inc.php, allerdings finden sich weitere Währungsfunktionen auch in kurspflege.php.

5. Auch die Anwender sind nicht ganz zufrieden. Manchmal ist die Datenbank nicht verfügbar. Dieser Fehler wird nicht abgefangen, anstelle dessen werden die Anwender mit unverständlichen Fehlermeldungen gequält.

Erste Gedanken zur Lösung

Glücklicherweise lassen sich diese Probleme leicht lösen. Probleme (1) und (2) lösen wir mit einem Anmeldeverfahren. Wir werden der Kurspflege-Seite eine Anmeldung vorschalten.

Problem (3) lösen wir, indem wir Funktionen schreiben, die die von uns genutzten MySql-Funktionen kapseln. Diese sammeln wir in einer Datei. Bei einer Änderung der Datenbank muss dann nur noch

in dieser Datei geändert werden. Mit dieser Vorgehensweise werden wir auch Problem (5) lösen. Es bietet sich nämlich an, hier zu prüfen, ob die Abfragen an die Datenbank auch erfolgreich waren.

Problem (4) lösen wir, indem wir alles, was mit Währung zu tun hat, in der Datei euroDollarUmrechnung.inc.php sammeln. Die dort geschriebenen Funktionen werden dann von den Seiten, die der Anwender sieht, aufgerufen (so funktioniert es ja bereits in produkte.php).

Anmeldung und Sessions

Doch beginnen wir mit dem Anmeldeverfahren[15]. Anmeldungen sind ein grundsätzliches Problem bei Internet-Anwendungen. Der Grund ist, dass das dem WWW zu Grundeliegende http-Protokoll "zustandslos" ist. Das bedeutet, dass ein WWW-Server immer genau eine Anfrage bearbeitet und die angeforderte WWW-Seite ausliefert (sofern diese Seite existiert). Danach "vergisst" der WWW-Server den Vorgang. Das würde bedeuten, die Anwender melden sich an, unsere Anwendung akzeptiert die Anmeldung und vergisst den Vorgang gleich wieder. Das wäre gar nicht gut. Es gibt allerdings zwei Möglichkeiten, diesem Zustand abzuhelfen:

- Man kann die Anmeldeinformation in einen Cookie schreiben. So ein Cookie wird ja, wie Sie wissen, bei jeder erneuten Anfrage an unseren WWW-Server vom Browser an den Server übertragen. Cookies lassen sich von php auswerten und so können wir überprüfen, ob der Benutzer bereits angemeldet ist oder nicht.

- Man kann die in jeder Seite referenzierten URL's verändern, indem man die Anmeldeinformation mit in die URL aufnimmt (URL-Rewriting). Die URL kann selbstverständlich auch von php aus ausgewertet werden.

Wenn man Anmeldeinformationen über mehrere Seiten einer Internetanwendung vorhält, spricht man von einer "Session".

In php ist das Erzeugen einer Session relativ einfach. Wir müssen in unseren Programmen Cookies nicht selber auswerten oder schreiben, oder Url's ändern. php hält für diese Zwecke Funktionen vor. Doch bevor wir anfangen, Sessions zu programmieren, müssen wir andere Dinge bedenken. Denn die Benutzer aus der Produktpflege-Abteilung müssen die Möglichkeit bekommen, sich an unserem System anzumelden. Vorher wollen wir sicherlich keine Session erzeugen.

Um dies abzubilden, müssen wir das Entity-Relationship-Modell ändern. Es bietet sich nämlich an, die Anmeldeinformationen (also die benötigten Informationen über die Mitarbeiter) ebenfalls in der Datenbank vorzuhalten.

Hier wird eine neue Entity und damit die neue Tabelle Mitarbeiter fällig (vgl. Abb. 10.12).

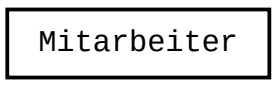


Abbildung 10.12: Das ERM zur Euro-Dollar-Problemstellung, die Entity Mitarbeiter

Wir könnten überlegen, ob die in Abb. 10.4 und Abb. 10.12 dargestellten Entitäten in einer Beziehung stehen (z.B. "darf ändern"). Wir halten das Beispiel jedoch einfach und sagen: Jeder, der sich anmelden kann, darf auch die Währungskurse ändern.

Dann müssen wir uns nur noch die Attribute überlegen. Hier besinnen wir uns auch auf das Notwendigste: mitarbeiter_nr, name, vorname, benutzerName, passwort. "mitarbeiter_nr" ist der selbst

[15] Das ist nämlich auch das schwierigste und das machen wir solange wir noch fit sind!

hoch zählende Primärschlüssel. Sie können sich selbst überlegen, aus welchem Grund wir das Feld "benutzerName" benötigen. Abb.10.13 zeigt die mit phpmyadmin neu angelegte Tabelle:

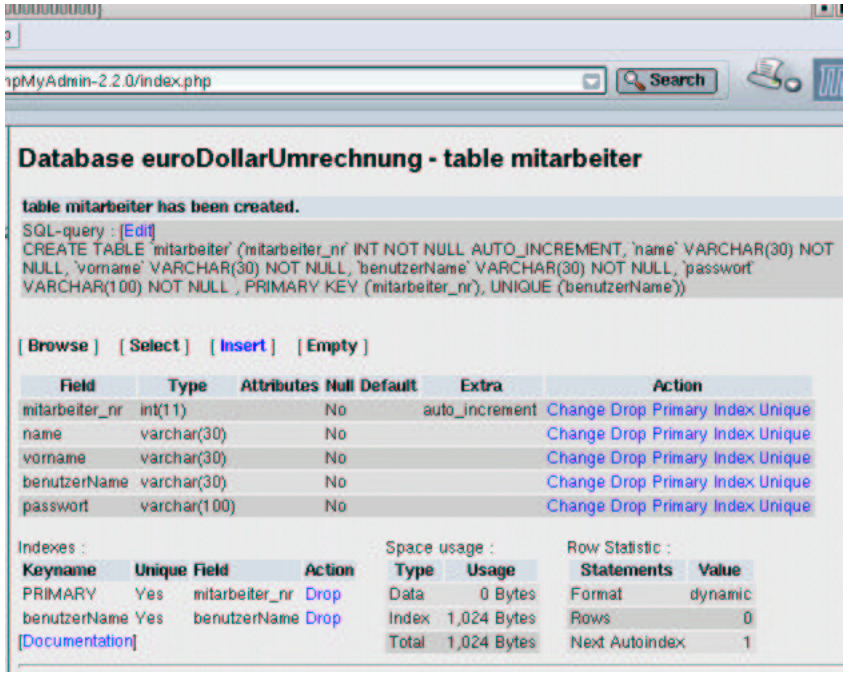


Abbildung 10.13: Die Tabelle Mitarbeiter

Bevor wir mit der Programmierung beginnen, stelle ich kurz die in php integrierte Session-Verwaltung vor. Um unsere Anwendung zu implementieren, benötigen wir folgende php-Funktionen:

- session_start(): Startet eine Session.
- session_register(): Registriert eine Variable in einer Session.
- session_is_registered(): Prüft, ob eine Variable in einer Session registriert wurde.
- session_destroy(): Zerstört eine Session.

session_start() startet, wie der Name schon sagt, eine Session. session_start() muss als erste Anweisung einer html-Seite ausgeführt werden und zwar bevor der Server Output an den Browser schickt, da session_start() header-Informationen verändert. Ein korrekter Aufruf von session_start() in einer html-Datei sieht also folgendermaßen aus:

```
<?php
        session_start();
?>
<html>
```

Beachten Sie, dass nicht einmal ein Leerzeichen vor dem <-Zeichen stehen darf, da der www-Server dann dieses an den Browser sendet. Um das tun zu können, muss der www-Server vorher die vollständigen header-Informationen senden und session_start() kann sie nicht mehr verändern.

session_start() checkt als Erstes, ob es bereits eine Session gibt. Ist dies nicht der Fall (dies betrachten wir zunächst), vergibt session_start() eine Session-Id. Dies ist eine für den Server eindeutige

Zufallszahl, die php dann noch mit dem md5-Algorithmus verschlüsselt. Diese Session-Id wird dann entweder (Cookies sind erlaubt) in einen Cookie geschrieben, oder alle Links werden umgeschrieben (URL-Rewriting). Abb. 10.14 zeigt dies am Beispiel unserer Intranet-Anwendung.

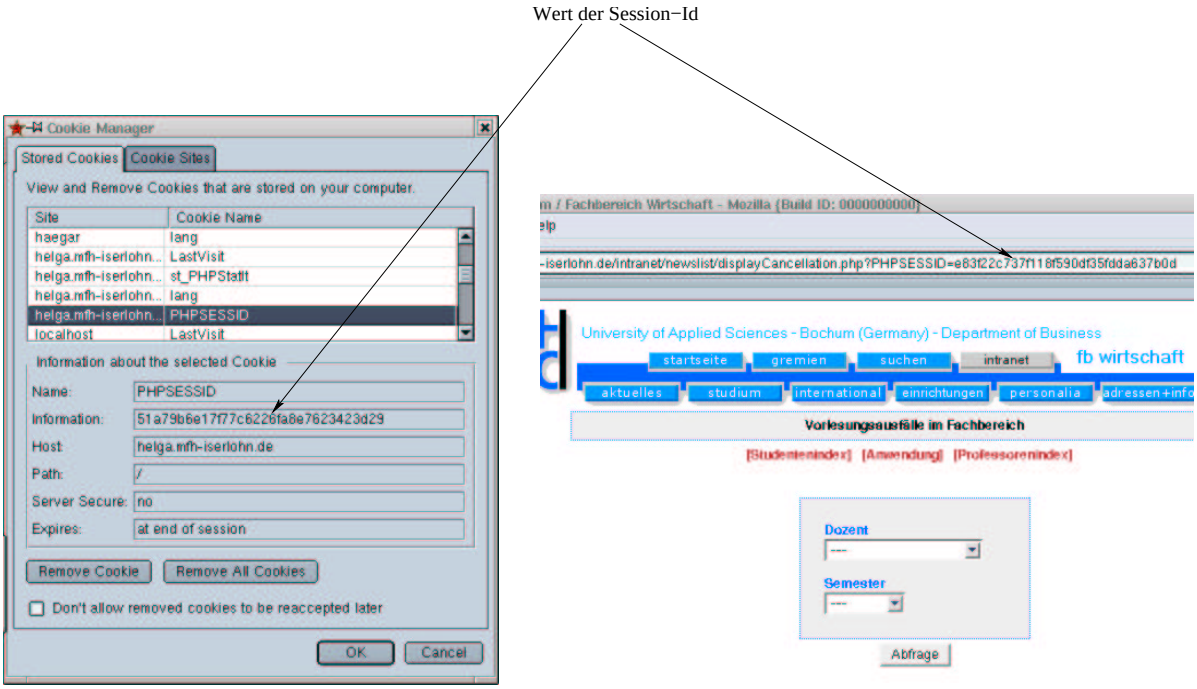


Abbildung 10.14: Session-Variablen als Cookie oder in der URL

Gibt es bereits eine Session, passiert gar nichts. Dies klingt zunächst nicht sonderlich intelligent, denn entweder gibt es keine Session, dann erzeugt session_start() eine Session, gibt es hingegen bereits eine, passiert nichts. Wie sollen wir also unterscheiden, ob ein Benutzer angemeldet ist oder nicht?

Hierzu stellt php die Funktionen session_register() und session_is_registered() zur Verfügung. session_register() registriert Variablen in einer Session, session_is_registered() überprüft, ob eine Variable in der aktuellen Session registriert ist.

Eine Variable für eine Session registrieren bedeutet, dass php in einem Verzeichnis für temporäre Dateien des Servers (/tmp in Unix/Linux) eine Datei mit dem Namen der vergebenen Session-Id anlegt und in diese Datei den Namen und den Wert der registrierten Variable einfügt. Abb. 10.15 zeigt dies. Unter Nutzung dieser beiden Funktionen erfolgt die Benutzerverwaltung dann in zwei Schritten:

1. Anmeldung:
    - Wir starten eine Session.
    - Die Benutzer geben Benutzernamen und Passwort an.
    - Wir überprüfen dies durch einen Datenbankzugriff.
    - Ist die Anmeldung okay, registrieren wir eine Session-Variable z.B. “benutzerName” und weisen dieser den Benutzernamen zu. Ist die Anmeldung nicht okay, zerstören wir die Session wieder (hierfür gibt es die Methode session_destroy).

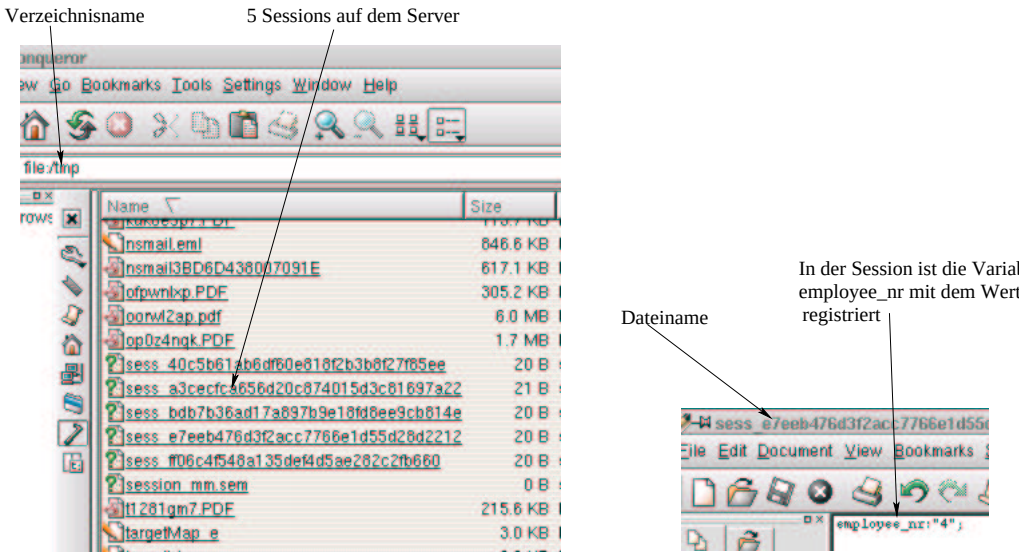


Abbildung 10.15: Variable employee_nr mit Wert 4 in einer Session

2. Überprüfung:

   - Wir starten eine Session.
   - Wir überprüfen, ob die Variable “benutzerName” registriert ist (mit session_is_registered()).
   - Ist dies der Fall, so ist der Benutzer angemeldet und darf die durch die Session geschützte Seite benutzen. Ist dies nicht der Fall, zerstören wir die Session wieder (wieder mit der Methode session_destroy) und verzweigen den Benutzer auf die Anmeldeseite.

Änderung der Anforderungen

Bevor wir mit der Implementierung beginnen, stimmen wir mit den Anwendern ab, wie die Anmeldung genau ablaufen soll. Hier bieten sich Aktivitätsdiagramme[16] an. Abb. 10.16 zeigt den Ablauf bei der Anmeldung.

Der Ablauf startet mit der Anmeldung des Benutzers. Ist die Anmeldung okay, wird eine Seite aufgerufen, die die Anwendungen referenziert, für die der Benutzer freigeschaltet ist. Ist die Anwendung nicht okay, wird der Benutzer auf die Anmeldungsseite zurückgeführt.

Gedanken zur Implementierung

Implementieren wir dies für unser Beispiel. Zunächst die Anmeldung: Als Erstes müssen wir eine Seite mit einem Formular aufblenden, in das der Benutzer die Anmeldeinformationen eingeben kann. Das ist nicht wirklich schwer, dazu reicht ein Formular mit einer Tabelle für das schöne Aussehen. Das Formular überprüfen wir mit JavaScript auf Eingabefehler (hier können wir nur überprüfen, ob überhaupt Eingaben getätigt wurden) und schicken es, wie gehabt, mit JavaScript ab.

Wenn das Formular abgeschickt wurde, müssen wir schon mehr machen:

1. Eine Verbindung zum Datenbank-System aufbauen und “euroDollarUmrechnung” als Datenbank einstellen.

[16] Ebenfalls Teil der UML.

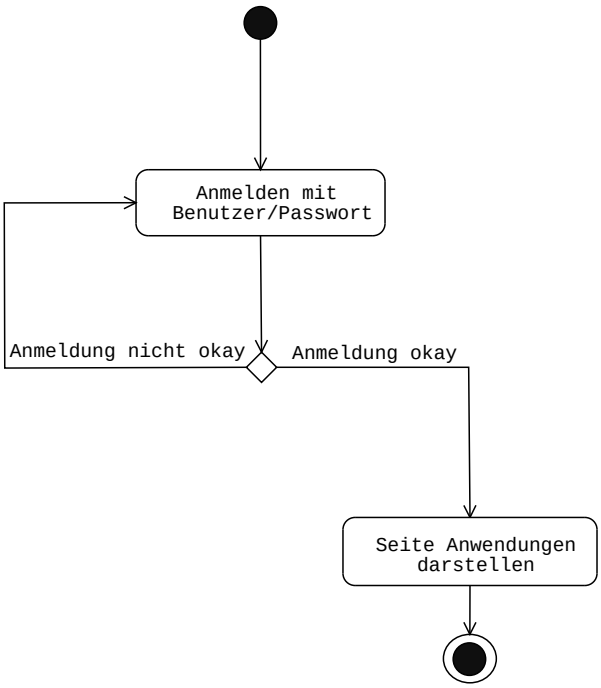


Abbildung 10.16: Ablauf bei der Anmeldung, Darstellung der Seiten anmeldung.php und anwendungen.php

2. Überprüfen, ob es einen Mitarbeiter mit der angegebenen Benutzernamen/Passwort-Kombination gibt.

3. Abhängig vom Erfolg dieses Vergleichs entweder auf die Seite mit den freigeschalteteten Anwendungen oder wieder zurück zur Anmeldeseite verzweigen.

Im Licht der Kritik an unserer Anwendung (Punkte (3) und (5)) bietet es sich an, die Verbindung zum Datenbanksystem und die Auswahl der Datenbank “euroDollarUmrechnung” in eine eigene Datei auszulagern. Wir erzeugen hierfür eine Klasse (vgl. 8), die wir DbFunctions nennen und in der wir alle Datenbankaufrufe durchführen werden. Sollte sich irgendwann einmal das zu Grundeliegende Datenbanksystem ändern, müssen wir nur in dieser Klasse Änderungen durchführen.

Wenn wir Punkt (2) betrachten, fällt uns auf, dass das Feststellen, ob eine Benutzer/Passwort-Kombination übereinstimmt, eigentlich eine Funktionalität der Mitarbeiter ist. Im Kritikpunkt (4) an unserer bisherigen Anwendung, wurde angemerkt, dass wir Funktionalitäten, die eigentlich Währungen zukommen, in anderen Dateien realisiert haben. Um hier vorzubeugen, werden wir eine Klasse Mitarbeiter erzeugen und dort alle Mitarbeiter betreffenden Funktionalitäten sammeln.

Beachten Sie, dass wir die Klassen in diesem Beispiel nicht erzeugen, um wirklich objektorientiert zu entwickeln. Wie werden keine Objekte dieser Klassen erzeugen. Bezugnehmend auf meine Argumentation in Kapitel 8 erstelle ich hier Klassen, um getrennte Namensräume zu erhalten und damit der Leser der Programme sofort sieht, in welcher Datei welche Funktion abgespeichert wird[17].

Bei Punkt (3) müssen wir eigentlich nur abhängig vom Ausgange der Überprüfung der Benutzernamen/Passwort-Kombination auf andere (oder die gleiche) html-Seiten verzweigen. Dies können wir (u.a.) durch die Nutzung der replace-Methode des JavaScript-location-Objekts erreichen. Die zugehörige JavaScript-Funktion müssen wir aus php erzeugen. Wie so etwas gemacht wird, haben Sie ja bereits in Kapitel

[17] Dies deshalb, weil wir unsere Klassen in Dateien mit dem Klassennamen abspeichern werden und dem Aufruf einer Klassenmethode ja der Klassenname vorangestellt wird.

10.3 gesehen. Hier lagern wir diese Funktion allerdings in eine eigene Datei aus. Und zur Verwirrung werden wir in diesem Fall keine Klasse erzeugen, sondern diese und ähnliche Funktionen in einer eigenen Datei sammeln. In dieser Datei werden wir alle Funktionen der Anwendung sammeln, die wir sonst nirgendwo zuordnen können (wo wir also keinen Oberbegriff und demzufolge auch keinen Klassennamen für finden können). Insgesamt ergibt sich der in Abb. 10.17 dargestellte Zusammenhang.

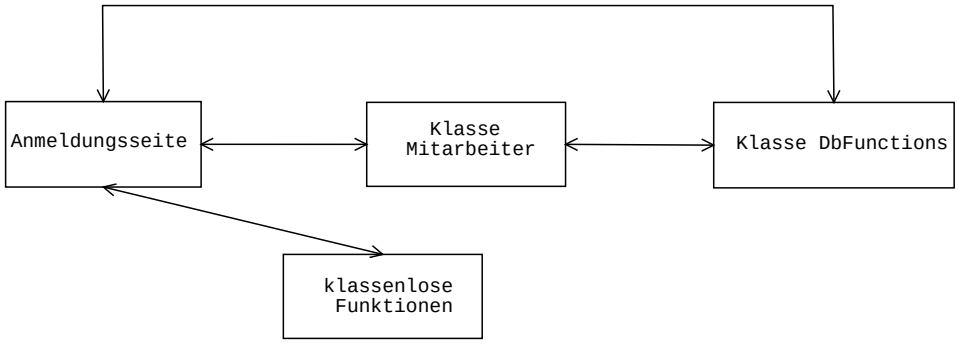


Abbildung 10.17: Architektur der Anwendung: Teil 1

Die Klasse DbFunctions

Beginnen wir mit der Klasse DbFunctions:

Beispiel 10.8 *Die Klasse DbFunctions*

```
<?php
class DbFunctions
{
/*
* class: DbFunctions
* Filename:  DbFunctions.inc.php
* Directory: classes
* lastChanged:  07.01.2002 by bb
* authors:  Bernd Bluemel, Christian Metzger
* purpose:  Common usable db-Functions
*/

// no instance-Variables, only classmethods, the class-construct is used to
 create a
// separate namespace for our DbFunctions

/*  boolean connect(&$link)
* connects to mysql, provides username and password
* creates the Link to the database
* the link is returned as the only parameter in
* the argument list
* sets the database to euroDollarUmrechnung
* returns true on success, false otherwise
*
*/
```

```
    function connect(&$link)
    {
        $link = mysql_pconnect("localhost", "root", "");
        if (!$link) return false;
        // select the database
        $query = "use euroDollarUmrechnung";
        mysql_query($query, $link);
        return true;
    }
    function getFirstFieldOfResult($link, $query)
    {
        $result=mysql_query($query, $link);
        if(mysql_num_rows($result)==0)
        {
            return;
        }
        return(mysql_result($result,0,0));
    }
    function executeQuery($link, $query)
    {
        $result=mysql_query($query, $link);
        return $result;
    }
}
?>
```

Wir betrachten zunächst die Funktion connect():

```
    function connect(&$link)
    {
        $link = mysql_pconnect("localhost", "root", "");
        if (!$link) return false;
        // select the database
        $query = "use euroDollarUmrechnung";
        mysql_query($query, $link);
        return true;
    }
```

connect() erwartet einen Parameter ($link). An dem &-Zeichen erkennen wir, dass es sich um einen Referenzparameter handelt. Änderungen dieses Parameters werden also an das aufrufende Programm zurückgegeben. connect() stellt also eine Verbindung zum Datenbank-System her und wählt die Datenbank “euroDollarUmrechnung” aus. Auf dem Übergabeparameter $link wird die Verbindung zur Datenbank an das aufrufende Programm zurückgegeben. Die Funktion gibt allerdings noch eine zweite Information an das aufrufende Programm zurück: Funktioniert der Aufbau der Verbindung zur Datenbank nicht ($link hat dann keinen Wert), gibt die Funktion false zurück. Im Erfolgsfall wird true zurückgegeben.

Die zweite Funktion der Klasse DbFunctions ist getFirstFieldOfResult():

```
    function getFirstFieldOfResult($link, $query)
    {
        $result=mysql_query($query, $link);
```

```
        if(mysql_num_rows($result)==0)
        {
            return;
        }
        return(mysql_result($result,0,0));
    }
```

Diese Funktion erwartet zwei Übergabeparameter, die Verbindung zur Datenbank ($link) und eine SQL-Abfrage. Die Funktion führt die SQL-Abfrage durch und gibt entweder das erste Feld in der ersten Zeile der Resultatsmenge zurück oder nichts. getFirstFieldOfResult() benutzt eine neue php-Funktion: mysql_num_rows(). mysql_num_rows() benötigt die Ergebnismenge einer SQL-Abfrage als Übergabeparameter. mysql_num_rows() gibt[18] die Anzahl der gefundenen Datensätze[19] zurück.

Die letzte Funktion von DbFunctions ist executeQuery():

```
    function executeQuery($link, $query)
    {
        $result=mysql_query($query, $link);
        return $result;
    }
```

executeQuery() gibt einfach das Ergebnis einer an die MySql-Datenbank gerichteten SQL-Abfrage zurück.

Die Klasse Mitarbeiter

In der Klasse Mitarbeiter benötigen wir zur Zeit nur eine Funktion, nämlich die, die überprüft, ob eine Benutzernamen/Passwort-Kombination okay ist:

Beispiel 10.9 *Die Klasse Mitarbeiter*

```
<?php
    // require_once("DbFunctions.inc.php");
class Mitarbeiter
{
/*
* class: Mitarbeiter
* Filename:  Mitarbeiter.inc.php
* Directory: classes
* lastChanged:  07.01.2002 by bb
* authors:  Bernd Bluemel, Christian Metzger
* purpose:  User related functions
*/
    function passwortIstOkay($link, $name, $passwort)
    {
        $query= "select * from mitarbeiter where ";
        $query.="benutzerName='$name' and ";
        $query.="passwort='$passwort'";
        $mitarbeiter_nr=DbFunctions::getFirstFieldOfResult($link, $query);
        if(!isset($mitarbeiter_nr) || empty ($mitarbeiter_nr))
        {
```

[18] Wie der Name schon andeutet.

[19] Dies sind die Zeilen der Ergebnismenge.

```
            return false;
        }
        return true;
    }
}
?>
```

passwortIstOkay() erwartet drei Übergabeparameter: Die Verbindung zur Datenbank ($link), den Benutzernamen ($name) und das Passwort ($passwort). passwortIstOkay() erzeugt dann ein SQL-Kommando, um festzustellen, ob es die Benutzernamen/Passwort-Kombination gibt:

```
$query= "select * from mitarbeiter where ";
$query.="benutzerName='$name' and ";
$query.="passwort='$passwort'";
```

Die Ergebnismenge zu diesem SQL-Kommando besteht entweder aus einem oder aus keinem Datensatz. Ist die Ergebnismenge leer, gibt es die Benutzernamen/Passwort-Kombination nicht und die Anmeldung ist fehlgeschlagen. passwortIstOkay() ruft dann getFirstFieldOfResult() auf. Sie sehen, dass dem Methodenaufruf bei der Programmierung mit Klassen der Klassennamen gefolgt von zwei Doppelpunkten vorangestellt werden muss. getFirstFieldOfResult() hatten wir weiter oben bereits besprochen. getFirstFieldOfResult() gibt nichts zurück, wenn die Ergebnismenge des SQL-Kommandos leer war, das erste Feld des ersten[20] Datensatzes der Ergebnismenge ansonsten. Wir prüfen also nur noch, ob das Ergebnis von getFirstFieldOfResult() existiert oder nicht. Existiert es, war die Anmeldung okay und es wird true zurückgegeben, existiert es nicht, wird false zurückgegeben. Zur Überprüfung benutzen wir die von php bereitgestellten Funktionen isset und empty. isset gibt true zurück, wenn eine Variable existiert, empty, wenn eine Variable keinen Wert hat.

Die von php erzeugten JavaScript-Funktionen

Beispiel 10.10 *Oft benutzte Funktionen*

```
<?php
// oft benutzte Funktionen
// in einer Datei zusammengefasst
// keine Klasse, wegen Schreibersparnis
// Dateiname:oftBenutzteFunktionen.inc.php
// Verzeichnis: includes
// Autor: Bernd Bluemel
// letzte Aenderung 07.01.2002 by bb

function loadPage($newPage)
{

echo "<script language=\"JavaScript\">";
echo "location.replace(\"$newPage\")";
echo "</script>";
}
function displayAlert ($alertMessage)
{
```

[20] Und im Fall unserer Abfrage auch einzigen

```
echo "<script language=\"JavaScript\">";
echo "alert(\"$alertMessage\")";
echo "</script>";
}
?>
```

Diese Funktionen müssten Sie eigentlich sofort verstehen. Die Funktion loadPage() lädt die ihr übergebene URL in das Browser-Fenster, displayAlert() gibt den ihr übergebenen Text in einem Alert-Fenster auf. Beachten Sie das “Escapen” der Sonderzeichen in den beiden Funktionen.

Die Datei anmeldung.php

Implementieren wir nun die Anmeldeseite:

Beispiel 10.11 *Die Anmeldeseite*

```
<?php
    session_start();
    require_once("./classes/Mitarbeiter.inc.php");
    require_once("./classes/DbFunctions.inc.php");
    require_once("./includes/oftBenutzteFunktionen.inc.php");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    anmeldung
  Dateiname: grosseGanze2/anmeldung.php //-->
<html>
<head>
  <title> Anmeldung </title>
    <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
    </script>
    <script language = "JavaScript">
        function teste()
        {
            if(istLeer(document.anmeldung.name,
                            "Kein Name eingegeben"))
            {
                return;
            }
            if(istLeer(document.anmeldung.passwort,
                            "Kein Passwort eingegeben"))
            {
                return;
            }
            document.anmeldung.method="post";
            document.anmeldung.action=document.anmeldung.action.value;
            document.anmeldung.submit();
        }
    </script>
```

```
</head>
<body>
    <table border=2 align="center">
             <tr>
               <td>
                    <img src="images/swan.jpg" width=96 height=80
                                                align="left" border=1>
               </td>
               <td>
                    Die Schwanen Gesellschaft
               </td>
             </tr>
    </table>
<?php
    if($REQUEST_METHOD!="POST")
    {
?>
        <h2 align="center">
            Anmeldung
        </h2>
        <hr>
        <form name="anmeldung">
            <input type="hidden" name="action"
                value="<?php echo $PHP_SELF ?>" >
            <table border=2 align="center">
                <tr>
                <td>
                        Benutzername:
                </td>
                <td>
                        <input type="text" name="name" size="12">
                </td>
                </tr>
                <tr>
                <td>
                        Passwort:
                </td>
                <td>
                        <input type="text" name="passwort" size="12">
                </td>
                </tr>
                </tr>
                    <td colspan="2" align="center">
                        <input type="button" value="Anmelden" onClick="teste()">
                    </td>
            </table>
        </form>
<?php
    }
    else
    {
        if(!DbFunctions::connect($link))
        {
            echo "Fehler";
```

```
            echo "</body>";
            echo "</html>";
            exit();
        }
        if(Mitarbeiter::passwortIstOkay($link, $name, $passwort))
        {
            $benutzerName=$name;
            session_register("benutzerName");
            loadPage("anwendungen.php");
        }
        else
        {
            displayAlert("Passwort-Benutzernamen-Kombination stimmt nicht!");
            loadPage($PHP_SELF);
            exit;
        }
    }
?>
</body>
</html>
```

Als erste Anweisung starten wir eine Session. Danach importieren wir die von anmeldung.php benötigten Dateien:

```
<?php
    session_start();
    require_once("./classes/Mitarbeiter.inc.php");
    require_once("./classes/DbFunctions.inc.php");
    require_once("./includes/oftBenutzteFunktionen.inc.php");
?>
```

Den Aufbau des Formulars, die JavaScript-Eingabeüberprüfungen und die Methode festzustellen, ob die php-Datei das erste Mal aufgerufen wurde oder als Folge des Abschickens des Formulars, dies alles kennen Sie bereits, so dass ich auf eine erneute Diskussion verzichte.

Wenn anmeldung.php als Folge der Übertragung des Formulars aufgerufen wird, wird zunächst die Verbindung zur Datenbank aufgebaut:

```
    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
        echo "</html>";
        exit();
    }
```

Funktioniert das, ist die Verbindung auf der Variablen $link abgespeichert, funktioniert das nicht, gibt es eine Fehlermeldung und die Anwendung beendet sich.

Nach der Erstellung der Verbindung zur Datenbank wird die Methode passwortIstOkay() der Mitarbeiterklasse genutzt.

```
    if(Mitarbeiter::passwortIstOkay($link, $name, $passwort))
    {
        $benutzerName=$name;
        session_register("benutzerName");
        loadPage("anwendungen.php");
    }
    else
    {
        displayAlert("Passwort-Benutzernamen-Kombination stimmt nicht!");
        loadPage($PHP_SELF);
        exit;
    }
```

War die Anmeldung erfolgreich, wird der Benutzername als Session-Variable registriert:

```
    session\_register("benutzerName")
```

Die Seite mit den erlaubten Anwendungen wird geladen (loadPage(“anwendungen.php”)). Bei erfolgloser Anmeldung wird die dementsprechende Fehlermeldung dargestellt und die Seite ruft sich selber wieder auf, um eine Neueingabe der Anmeldedaten zu gestatten (loadPage($PHP_SELF)).

Abb. 10.18 zeigt dies anhand der Abläufe im Browser. Beachten Sie die Ähnlichkeit mit Abb. 10.16.

Die Datei anwendungen.php

Diskutieren wir nun kurz die Seite anwendungen.php:

Beispiel 10.12 *Die Seite Anwendungen*

```
<?php
    session_start();
    require_once("./classes/Mitarbeiter.inc.php");
    require_once("./classes/DbFunctions.inc.php");
    require_once("./includes/oftBenutzteFunktionen.inc.php");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    anwendungen
  Dateiname: grosseGanze2/anwendungen.php //-->
<html>
<head>
  <title> Anwendungen </title>
</head>
<body>
    <table border=2 align="center">
            <tr>
              <td>
                   <img src="images/swan.jpg" width=96 height=80
                                              align="left" border=1>
              </td>
              <td>
                   Die Schwanen Gesellschaft
```

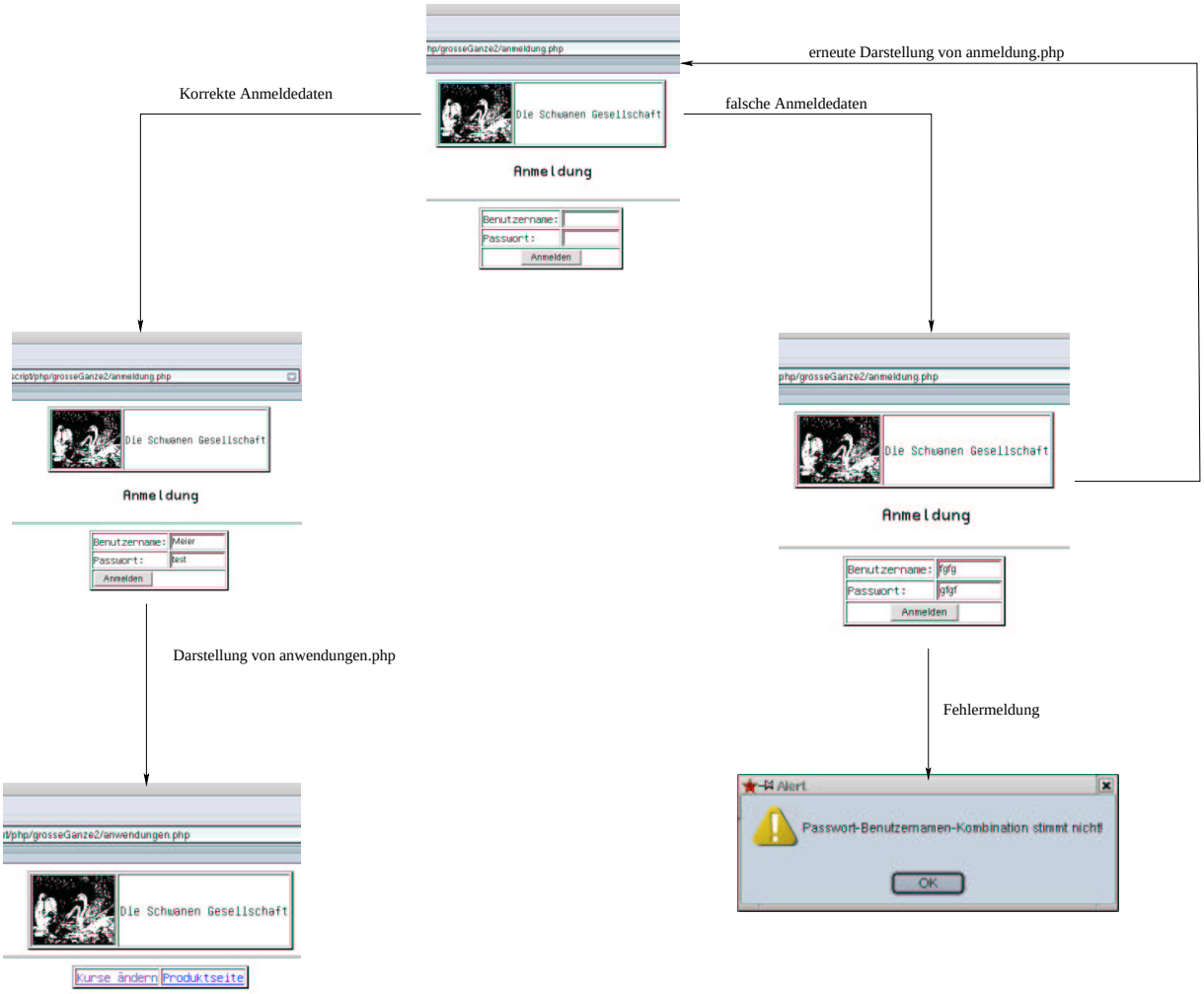


Abbildung 10.18: Ablauf bei der Anmeldung im Browser

```
                </td>
              </tr>
    </table>
    <hr>
<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>
    <table border=2 align="center">
    <tr>
        <td> <a href="./kurspflege.php"> Kurse &auml;ndern </a> </td>
        <td> <a href="./produkte.php"> Produktseite </a> </td>
    </tr>
```

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

Da diese Seite vor unbefugten Benutzern geschützt werden soll, wird auch hier zunächst eine Session gestartet:

```
<?php
    session_start();
```

Die Überprüfung, ob ein Benutzer angemeldet ist, und damit die Seite benutzen darf, findet durch folgende Zeilen statt:

```
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
```

Wenn die Variable benutzerName innerhalb der Session registriert ist, werden obige Zeilen nicht ausgeführt. Dies ist der Fall, wenn der Anwender sich erfolgreich angemeldet hat, denn dann wurde ja durch die Seite anwendungen.php eben diese Variable für die Session registriert. Ist die Variable jedoch nicht registriert (und das bedeutet, der Benutzer ist nicht angemeldet), wird in das `if`-Konstrukt verzweigt (beachten Sie das Ausrufungszeichen in der Bedingung des `if`-Konstrukts). Dort wird zunächst eine Fehlermeldung ausgegeben (displayAlert(“Nicht angemeldet”)), dann wird die Anmeldeseite geladen (loadPage(“anmeldung.php”), um dem Benutzer die Chance zu geben, sich anzumelden.

Die Datei kurspflege.php und die Währungsdatei

Bevor wir uns die Seite kurspflege.php ansehen, müssen wir uns vorab einige Gedanken machen. Da wir die Anforderung (4) –alle Währungsfunktionen in einer eigenen Datei– realisieren müssen, müssen wir, bevor wir kurspflege.php selber realisieren, die Datei euroDollarUmrechnung.inc.php ändern. Denn dort müssen wir die Kursänderung (die diesbezüglichen Anweisungen stehen zur Zeit ja noch in kurspflege.php) realisieren. Darüber hinaus benennen wir die Datei um: Da wir hier alle unsere Währungsfunktionen sammeln, erzeugen wir eine neue Klasse und nennen diese Klasse Währung. Das macht es uns später auch vom Namen her einfacher, weitere Währungen zu integrieren. Desweiteren nehmen wir den Verbindungsaufbau zur Datenbank aus dieser Klasse heraus, dafür haben wir die Klasse DbFunctions erzeugt.

Damit sieht die neue Klasse Waehrung.inc.php folgendermaßen aus:

Beispiel 10.13 *Die Klasse Wahrung*

```
<?php
require_once("./classes/DbFunctions.inc.php");
class Waehrung
{
/*
* class: Waehrung
```

```
* Filename:  Waehrung.inc.php
* Directory: classes
* lastChanged:  07.01.2002 by bb
* authors:  Bernd Bluemel
* purpose:        Funktionen zur Dollar-Euro
*                       oder Euro-Dollar Umrechnung
*/
    function holeDollarKurs($link)
    {
        $query="Select kurs from waehrung where name='Dollar'";
        $kurs=DbFunctions::getFirstFieldOfResult($link, $query);
        return $kurs;
    }
    function euroDollarUmrechnung($link, $zielwaehrung, $betrag)
    {
        $kurs=Waehrung::holeDollarKurs($link);
        if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
        {
            $dollarbetrag=$kurs*$betrag;
            return $dollarbetrag;
        }
        if(($zielwaehrung=="Euro")||($zielwaehrung=="euro"))
        {
            $eurobetrag=(1/$kurs)*$betrag;
            return  $eurobetrag;
        }
    }
    function erzeugeEuroDollarUmrechnungsScript($link)
    {
        $kurs=Waehrung::holeDollarKurs($link);
        echo "<script language=\"JavaScript\">\n";
            echo "function euroDollarUmrechnung(zielwaehrung, betrag)\n";
            echo "{\n";
                echo "var kurs=$kurs;\n";
                echo "if((zielwaehrung==\"Dollar\")||(zielwaehrung==\"dollar\"))\n";
                echo "{\n";
                    echo "dollarbetrag=kurs*betrag;\n";
                    echo "return(dollarbetrag);\n";
                echo "}\n";
                echo "if((zielwaehrung==\"Euro\")||(zielwaehrung==\"euro\"))\n";
                echo "{\n";
                    echo "eurobetrag=(1/kurs)*betrag;\n";
                    echo "return(eurobetrag);\n";
                echo "}\n";
            echo "}\n";
        echo "</script>\n";
    }

    function aendereDollarKurs($link, $kurs)
    {
        $updateQuery ="update waehrung set kurs='$kurs' ";
        $updateQuery.="where name='Dollar'";
        DbFunctions::executeQuery($link, $updateQuery);
        return;
```

```
    }
}
?>
```

Hier ist eigentlich nur die Funktion aendereDollarKurs() neu. Sie erwartet zwei Parameter: $link und $kurs. Auf $link ist, wie immer, die Verbindung zur Datenbank abgespeichert, $kurs enthält den neuen Dollarkurs. Die Funktion an sich enthält nichts Neues. Wir haben nur die bislang in kurspflege.php beheimateten Befehle in unsere Währungsklasse ausgelagert.

Die weitere Veränderung gegenüber der alten Datei euroDollarUmrechnung.inc.php besteht nun darin, dass wir eine Klasse erzeugt haben und jeder Funktion dieser neuen Klasse die Verbindung zur Datenbank ($link) zusätzlich übergeben. Die Nutzung von Waehrung.inc.php setzt also eine bestehende Verbindung zur Datenbank voraus.

Betrachten wir nun die Seite kurspflege.php:

Beispiel 10.14 *Die Seite Kurspflege*

```
<?php
    session_start();
    require_once("./classes/DbFunctions.inc.php");
    require_once("./classes/Waehrung.inc.php");
    require_once("./includes/oftBenutzteFunktionen.inc.php");
?>
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Kurspflege
  Dateiname: grosseGanze2/kurspflege.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung Kurspflege</title>
  <script language = "JavaScript"
           src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript"
           src="./javascript/stringbearbeitung.js">
  </script>

  <script language = "JavaScript">
    function teste()
    {
        document.kurspflege.kurs.value=komma2punkt(
                                          document.kurspflege.kurs.value);

        if(istKeineZahl(document.kurspflege.kurs,
                        "Kurs ist keine Zahl"))
        {
            return false;
        }
        document.kurspflege.method="post";
        document.kurspflege.action=document.kurspflege.action.value;
        document.kurspflege.submit();
    }
```

```
  </script>
</head>
<body>
<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>
    <table border=2 align="center">
             <tr>
               <td>
                    <img src="images/swan.jpg" width=96 height=80
                                                 align="left" border=1>
               </td>
               <td>
                    Die Schwanen Gesellschaft
               </td>
             </tr>
    </table>
    <h2 align="center">
        Kurspflege
    </h2>
    <hr>
<?php
    // Wir pruefen ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
?>
        <form name='kurspflege'>
            <input type="hidden" name="action"
                        value="<?php echo $PHP_SELF ?>" >
            <table border align="center">
                <tr>
                    <td>
                        Neuer Kurs
                    </td>
                    <td>
                        <input type="text" name="kurs" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="button" name="Button1" onClick="teste()"
                                                          value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
```

```
    else
    {
        if(!DbFunctions::connect($link))
        {
            echo "Fehler";
            echo "</body>";
            echo "</html>";
            exit();
        }
        // update sql-Kommando
        Waehrung::aendereDollarKurs($link, $kurs);
        echo "<p align=\"center\">";
        echo "Kurs ge&auml;ndert auf $kurs.";
        echo "</p>";
    }
?>
</body>
</html>
```

Neu ist zunächst das Starten der Session und der Import der Dateien, deren Funktionen wir nutzen wollen:

```
<?php
    session_start();
    require_once("./classes/DbFunctions.inc.php");
    require_once("./classes/Waehrung.inc.php");
    require_once("./includes/oftBenutzteFunktionen.inc.php");
?>
```

Wie bei anwendungen.php überprüfen wir dann im Body als Erstes, ob die Seite von einem angemeldeten Benutzer aufgerufen wurde:

```
<?php
    if(!session_is_registered("benutzerName"))
    {
        displayAlert("Nicht angemeldet");
        loadPage("anmeldung.php");
        exit();
    }
?>
```

Auch hier wird verhindert, dass nicht angemeldete Benutzer die Seite nutzen dürfen. Nicht angemeldete Benutzer werden auf die Anmeldeseite zurückgeleitet.

Die nächste Änderung ergibt sich im `else`-Teil. Da die neue Klasse Waehrung eine Verbindung zur Datenbank voraussetzt, schaffen wir diese:

```
    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
```

```
        echo "</html>";
        exit();
    }
```

Diese Zeilen sind uns ja bereits bekannt. Wir bauen mit der Methode connect() der Datenbankklasse DbFunctions eine Verbindung zur Datenbank auf. Funktioniert dies, ist die Verbindung auf $link abgespeichert, ansonsten gibt es eine Fehlermeldung und das Programm wird beendet.

Nun bleibt nur noch die in Waehrung implementierte Methode aendereDollarKurs() aufzurufen:

```
Waehrung::aendereDollarKurs($link, $kurs);
```

Ein Screenshot dieser Seite erübrigt sich, da sich in der Bildschirmdarstellung gegenüber Kapitel 10.3 keine Änderungen ergeben haben.

Die Datei produkte.php

Hier gibt es die wenigsten Änderungen. Diese Seite darf nicht vor Zugriffen der Benutzer geschützt werden. Schließlich sollen die ja was bestellen und da ist es eigentlich unsinnig, den Zugriff auf diese Seite zu reglementieren. Der einzige Unterschied zur Implementierung in Kapitel 10.3 besteht darin, dass wir die Verbindung zur Datenbank unter Zuhilfenahme von DbFunktions in produkte.php selber aufbauen und dann die Verbindung mit Hilfe der Variablen $link an die aufgerufenen Methoden in der Waehrungsklasse übergeben. darüber hinaus müssen wir den Klassennamen (Waehrung) gefolgt von zwei Doppelpunkten vor den Methodenaufruf schreiben. Aber das kennen wir ja schon von den anderen Beispielen in diesem Kapitel. Damit ergibt sich als neue Implementierung von produkte.php:

Beispiel 10.15 *Die Seite Produkte*

```
<!-- Programm zur Euro-Dollar Umrechnung
    Teil Das grosse Ganze
    Produktseite mit Euro-Rechner
  Dateiname: grosseGanze2/produkte.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Euro-Dollar Umrechnung: Produkte</title>
<?php
    require_once("./classes/Waehrung.inc.php");
    require_once("./classes/DbFunctions.inc.php");
    if(!DbFunctions::connect($link))
    {
        echo "Fehler";
        echo "</body>";
        echo "</html>";
        exit();
    }
    Waehrung::erzeugeEuroDollarUmrechnungsScript($link);
?>
  <script language = "JavaScript"
            src="./javascript/feldkontrolle.js">
  </script>
  <script language = "JavaScript"
            src="./javascript/stringbearbeitung2.js">
  </script>
```

```
  <script language = "JavaScript">
    function dollar2euro()
    {
            document.euro6.dollar.value=komma2punkt
                            (document.euro6.dollar.value);
            if(istKeineZahl(document.euro6.dollar,
                                "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            {
                document.euro6.euro.value=
                        schoeneAusgabe(euroDollarUmrechnung(
                            "euro", document.euro6.dollar.value).toString());
            }
    }
    function euro2dollar()
    {
            document.euro6.euro.value=komma2punkt
                            (document.euro6.euro.value);
            if(istKeineZahl(document.euro6.euro,
                                "Keine Zahl eingegeben!"))
            {
                return;
            }
            else
            {
                document.euro6.dollar.value=
                        schoeneAusgabe(euroDollarUmrechnung(
                            "dollar", document.euro6.euro.value).toString());
            }
    }
    </script>
</head>
<body>
    <table border=2 align="center">
             <tr>
               <td>
                    <img src="images/swan.jpg" width=96 height=80
                                                 align="left" border=1>
               </td>
               <td valign="bottom">
                    <form name='euro6'>
                        <table border>
                            <tr>
                                <td>
                                    Dollar
                                </td>
                                <td>
                                    <input type="text" name="dollar"
                                        onChange="dollar2euro()" size=12>
                                </td>
                            </tr>
```

```
                        <tr>
                            <td>
                                Euro
                            </td>
                            <td>
                                <input type="text" name="euro"
                                    onChange="euro2dollar()" size=12>
                            </td>
                        </tr>
                    </table>
                </form>
            </td>
        </tr>
</table>
<h2 align="center">
    Die Schwanen Gesellschaft: Unsere Produkte
</h2>
<hr>
<table border="2" align="center">
<tr>
    <th> Produktname </th>
    <th> Preis in Euro </th>
    <th> Preis in Dollar </th>
</tr>
<tr>
    <td> Tolle Seife </td>
    <td align="center"> 2,00 </td>
    <td align="center">
<?php
        echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                                                          2),2,","," "))
?>
    </td>
</tr>
<tr>
    <td> Tolle Zahnpasta </td>
    <td align="center"> 3,50 </td>
    <td align="center">
<?php
        echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                                                          3.5),2,","," "))
?>
    </td>
</tr>
<tr>
    <td> Zahnseide </td>
    <td align="center"> 4,50</td>
    <td align="center">
<?php
        echo(number_format(Waehrung::euroDollarUmrechnung($link,"Dollar",
                                                          4.5),2,","," "));
?>
    </td>
</tr>
```

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

Ein Screenshot dieser Seite erübrigt sich, da sich in der Bildschirmdarstellung gegenüber Kapitel 10.3 keine Änderungen ergeben haben.

Zusammenfassung

Auch hier zeigt sich, dass Internet-Anwendungen nicht ganz einfach zu realsieren sind, sondern Nachdenken und Grundkenntnisse in Programmierung erfordern. Moderne, auf Inter- oder Intranettechnologien basierende Informationssysteme sind nicht “einfach eben so” realisierbar, sondern erfordern Sachverstand und auch Entwicklungszeit. Abb. **??**eigt die die Dateistruktur der neuen Anwendung.

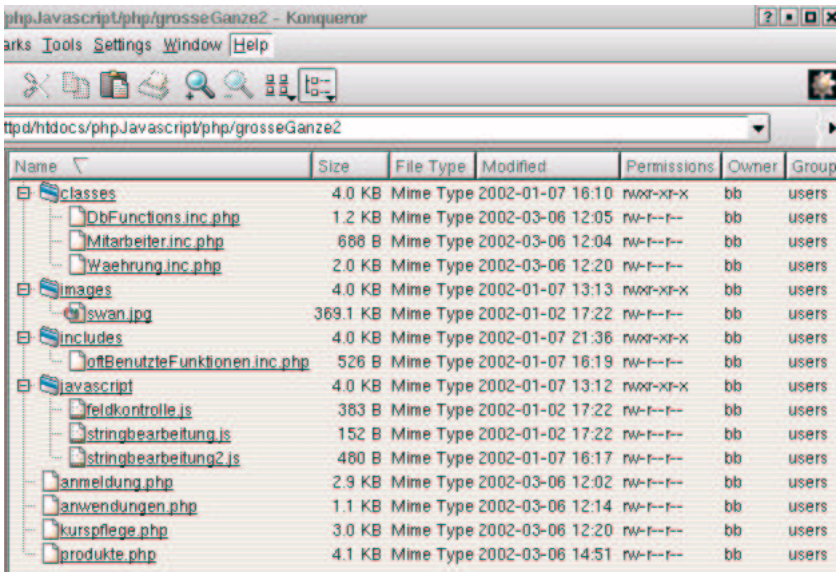


Abbildung 10.19: Dateistruktur der geänderten Anwendung

Kapitel 11

Lösungen

Lösungen Kapitel 2

Lösung Aufgabe 2.1

JavaScript

```
<!-- Das Programm gibt meinen Namen im Browser aus
  Dateiname: loesung2.1.html //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bernd Bl&uuml;mel</title>
</head>
<body>
    <script LANGUAGE = "JavaScript">
        document.write ("<H2> Bernd Bl&uuml;mel</H2>");
    </script>
</body>
</html>
```

php

```
<!-- Das Programm gibt meinen Namen im Browser aus
  Dateiname: helloWorld.php //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bernd Bl&uuml;mel</title>
</head>
<body>
<?php
        echo "<h2> Bernd Bl&uuml;mel </h2>";
?>
</body>
</html>
```

Lösung Aufgabe 2.2 Es gibt zwei noch viel einfachere Programme. Einmal gibt es sowohl in JavaScript, als auch in html die „leere“ Anweisung. Dies ist die Anweisung, die nur aus einem “;” besteht.

JavaScript

```
<!-- Das Programm gibt nichts im Browser aus
  Dateiname: loesung2.2.a.html //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Nichts</title>
</head>
<body>
    <script LANGUAGE = "JavaScript">
        ;
    </script>
</body>
</html>
```

php

```
<!-- Das Programm gibt nichts aus
  Dateiname: loesung2.2.a.php //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Nichts</title>
</head>
<body>
<?php
    ;
?>
</body>
</html>
```

Sodann kann man auch einfach gar keine Anweisung in ein Programm aufnehmen:

JavaScript

```
<!-- Das Programm gibt nichts im Browser aus
  Dateiname: loesung2.html //-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Nichts</title>
</head>
```

```
<body>
    <script LANGUAGE = "JavaScript">
    </script>
</body>
</html>
```

php

```
<!-- Das Programm gibt nichts aus
  Dateiname: loesung2.2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Nichts</title>
</head>
<body>
<?php
?>
</body>
</html>
```

Das Resultat ist beide Mal, dass nichts ausgegeben wird.

Lösung Aufgabe 2.3

JavaScript

```
<!-- Das Programm subtrahiert zwei einzugebende Zahlen
  Dateiname: loesung2.3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Subtraktion</title>
</head>
<body>
    <script language = "JavaScript">
        var subtrahend;
        var minuend;
        var differenz;
        subtrahend=prompt("Bitte geben Sie den Subtrahenden ein!","");
        minuend=prompt("Bitte geben Sie den Minuenden ein!","");
        subtrahend=parseFloat(subtrahend);
        minuend=parseFloat(minuend);
        differenz=subtrahend-minuend;
        document.write ("Die Differenz von " + subtrahend +
                                " und " + minuend +
                                " ist: " + differenz);
    </script>
</body>
</html>
```

php

```
<!-- Das Programm subtrahiert zwei Zahlen
  Dateiname: loesung2.3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Subtraktion</title>
</head>
<body>
<h2>
    Subtraktion zweier Zahlen
</h2>
<form name="subtraktion" action="./loesung2.3.php" method="post">
    <table border>
        <tr>
            <td>
                Subtrahend
            </td>
            <td>
                <input type="text" name="subtrahend" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Minuend
            </td>
            <td>
                <input type="text" name="minuend" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>


<!-- Das Programm subtrahiert 2 Zahlen
  Dateiname: loesung2.3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Subtraktion mit Eingabe in php (Teil 2)</title>
</head>
<body>
<h2>
    Subtraktion zweier Zahlen: Das Ergebnis
</h2>
<?php
    $differenz=$subtrahend-$minuend;
```

```
    echo ("Die Differenz von $subtrahend und $minuend" .
                    " ist: $differenz");
?>
</body>
</html>
```

Lösung Aufgabe 2.4

JavaScript

```
<!-- Das Programm addiert negative  natuerliche Zahlen
    bis zu einer eingegebenen Zahl
  Dateiname: loesung2.4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition negativer Zahlen</title>
</head>
<body>
    <script language = "JavaScript">
        var bisZu;
        var summe;
        bisZu=prompt("Bitte geben Sie die Zahl, bis zu der "+
                        " die ersten negativen Zahlen summiert werden " +
                        "sollen, ein!","");
        bisZu=parseInt(bisZu);
        summe=0
        for(i=1;i<=bisZu;i++)
        {
            summe=summe+i;
        }
        summe=-1*summe;
        document.write ("Die Summe der ersten " + bisZu +
                                " negativen nat&uuml;rlichen Zahlen " +
                                 " ist: " + summe);
    </script>
</body>
</html>
```

php

```
<!-- Das Programm summiert negative Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: loesung2.4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Aufsummierung aller negativen natürlichen Zahlen
                    bis zu einer einzugebender Zahl in php</title>
</head>
```

```
<body>
<h2>
    Aufsummierung aller negativen natürlichen Zahlen bis
    zu einer einzugebender Zahl
</h2>
<form name="addition3" action="./loesung2.4.php" method="POST">
    <table border>
        <tr>
            <td>
                Zahl, bis zu der summiert werden soll
            </td>
            <td>
                <input type="text" name="bisZu" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

```
<!-- Aufsummierung negativer ganzer Zahlen
  Dateiname: loesung2.4.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition negativer Zahlen (Teil 2)</title>
</head>
<body>
<h2>
    Aufsummierung negativer nat&uuml;rlicher Zahlen: Das Ergebnis
</h2>
<?php
        $summe=0;
        for($i=1;$i<=$bisZu;$i++)
        {
            $summe=$summe+$i;
        }
        $summe=-1*$summe;
        echo ("Die Summe der ersten $bisZu " .
                                " negativen nat&uuml;rlichen Zahlen " .
                                 " ist: $summe");
?>
</body>
</html>
```

Die Lösung mit der Formel verläuft vollständig analog zu Beispiel 2.13 bzw. bei-addition3Formelphp durch Einfügen der Zeile

```
summe=-1*summe;
```

vor dem `document.write`, bzw. der Zeile

```
$summe=-1*$summe;
```

vor dem `echo`.

Lösung Aufgabe 2.5 Code, den Sie geheimhalten wollen, können Sie nicht in JavaScript schreiben. Andererseits können Sie jedes von anderen in JavaScript geschriebene Programm, sofern es in einer Internet-Seite vorkommt, die Sie kennen, herunterkopieren und nutzen.

php-Code wird nicht an den Browser übertragen, kann daher von anderen auch nicht kopiert werden.

Was allerdings bei php auch nicht optimal ist, ist, dass wenn Sie für andere entwickeln, nicht wie bei Compiler-Sprachen ein Binary übergeben können, sondern den Quellcode (weil der ja schließlich vom http-Server ausgeführt wird). Um hier Abhilfe zu schaffen, gibt es von Zend eine kostenpflichtige php-Erweiterung, die den Code verschlüsselt.

Lösungen Kapitel 3

Lösung Aufgabe 3.1

JavaScript

```
<!-- Das Programm addiert negative  natuerliche Zahlen
    bis zu einer eingegebenen Zahl
  Dateiname: loesung3.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition negativer Zahlen</title>
</head>
<body>
    <script language = "JavaScript" src="./javascript/loesung3.1.js">
    </script>
</body>
</html>


// Javascript-code fuer loesung 3.1
// Dateiname: loesung3.1.js

    var bisZu;
    var summe;
    bisZu=prompt("Bitte geben Sie die Zahl, bis zu der "+
                    " die ersten negativen Zahlen summiert werden " +
                    "sollen, ein!","");
    bisZu=parseInt(bisZu);
    summe=0
    for(i=1;i<=bisZu;i++)

        summe=summe+i;

    summe=-1*summe;
    document.write ("Die Summe der ersten " + bisZu +
                         " negativen nat&uuml;rlichen Zahlen " +
                            " ist: " + summe);
```

php

```
<!-- Das Programm summiert negative Zahlen bis
    zu einer einzugebenden Zahl
  Dateiname: loesung3.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Aufsummierung aller
                negativen natürlichen Zahlen bis
                zu einer einzugebenden Zahl in php</title>
</head>
<body>
<h2>
```

```
    Aufsummierung aller negativen natürlichen Zahlen
    bis zu einer einzugebenden Zahl
</h2>
<form name="addition3" action="./loesung3.1.php" method="POST">
    <table border>
        <tr>
            <td>
                Zahl, bis zu der summiert werden soll
            </td>
            <td>
                <input type="text" name="bisZu" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>


<!-- Aufsummierung negativer nat&uuml;rlicher Zahlen:
    Das Ergebnis
    Dateiname: loesung3.1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Addition negativer Zahlen (Teil 2)</title>
</head>
<body>
<h2>
    Aufsummierung negativer nat&uuml;rlicher Zahlen: Das Ergebnis
</h2>
<?php
    require("./includes/loesung3.1.inc.php");
?>
</body>
</html>
```

```
<?php
    // php-Code für loesung 3.1
    // Dateiname: loesung3.1.inc.php
    $summe=0;
    for($i=1;$i<=$bisZu;$i++)
    {
        $summe=$summe+$i;
    }
    $summe=-1*$summe;
    echo ("Die Summe der ersten $bisZu " .
                        " negativen nat&uuml;rlichen Zahlen " .
                            " ist: $summe");
?>
```

Lösungen Kapitel 4

Lösung Aufgabe 4.1 Die Lösung hierzu schenke ich mir, das ist ja nun doch zu einfach :-)!

Lösung Aufgabe 4.2

Beispiel 4.6 in php

```
<!-- Das Programm zur Demonstration arithmetischer Operatoren
  Dateiname: loesung4.2a.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Arithmetische Operatoren</title>
</head>
<body>
<?php
    $i = 2;
    $j = 3;
    $r= 2;
    $s=3;
    $l=$i+$j; // 5;
    echo "l1: $l <br>";
    $l=$i-$j; // -1
    echo "l2: $l <br>";
    $l=$j%$i; // 1 (Modulo-Bildung)
    echo "l3: $l <br>";
    $l=$i*$j; // 6
    echo "l4: $l <br>";
    $t=$s/$r; // 1.5 Division
    echo "t: $t <br>";
    $l=$r+$i*$j; // 8
    echo "l4: $l <br>";
    $l=($r+$i)*$j;  // 12
    echo "l5: $l <br>";
?>
</body>
</html>
```

Beispiel 4.9 in php

```
<!-- Das Programm zur Demonstration von Vergleichsoperatoren
  Dateiname: loesung4.2b.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Vergleichsoperatoren</title>
</head>
<body>
<?php
        $s1="ab";
        $s2="cd";
        $i=5;
```

```
        $j="5";
        $ergebnis=($s1==$s2);
        echo "ergebnis1: $ergebnis <br>";
        $ergebnis=($s1=$s2);
        echo "ergebnis2: $ergebnis s1: $s1 <br>";
        $ergebnis=($s1!=$i);
        echo "ergebnis3: $ergebnis <br>";
        $ergebnis=($s1>$i);
        echo "ergebnis4: $ergebnis <br>";
        $ergebnis=($j==$i);
        echo "ergebnis5: $ergebnis <br>";
        $ergebnis=($j=$i);
        echo "ergebnis6: $ergebnis <br>";
?>
</body>
</html>
```

Beispiel 4.10 in php

```
<!-- Programm zur Demonstration von logischen Operatoren
  Dateiname: loesung4.2c.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Logische Operatoren</title>
</head>
<body>
<?php
        $i=5;
        $j=6;
        $k=6;
        $l=5;
        $ergebnis = ($i==$j) && ($j==$k); // Wert von $ergebnis: false, da $i nicht $
        echo "ergebnis1: $ergebnis <br>";
        $ergebnis = ($i==$j) || ($j==$k); // Wert von $ergebnis: true, da $j gleich $
        echo "ergebnis2: $ergebnis <br>";
        $ergebnis = ($i==$l) && ($j==$k); // Wert von $ergebnis: true
        echo "ergebnis3: $ergebnis <br>";
        $ergebnis = ($i==$l) || ($j==$k); // Wert von $ergebnis: true
        echo "ergebnis4: $ergebnis <br>";
        $ergebnis = !($i==$j);  // Wert von $ergebnis: true
        echo "ergebnis5: $ergebnis <br>";
        $ergebnis = !($i!=$j);  // Wert von $ergebnis: false
        echo "ergebnis6: $ergebnis <br>";
?>
</body>
</html>
```

Beispiel 4.11 in php

```
<!-- Programm zur Demonstration des tenaeren Operators
```

```
  Dateiname: loesung4.2d.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Ten&auml;rer Operator</title>
</head>
<body>
<?php
        $i = 1;
        $j = 2;
        ($i==$j)?$k=$i:$k=6;
        echo "k hat den Wert: $k";
?>
</body>
</html>
```

Lösung Aufgabe 4.3

And-Wertetabelle in JavaScript

```
<!-- Programm fuer die and-Wertetabelle
  Dateiname: loesung4.3a.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>And-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die And-Wertetabelle
    </p>
    <script LANGUAGE = "JavaScript">
        var a
        var b;
        var ergebnis;
        a=true;
        b=true;
        ergebnis=a&&b;
        document.write("true and true: " + ergebnis + "<br>");
        a=true;
        b=false;
        ergebnis=a&&b;
        document.write("true and false: " + ergebnis + "<br>");
        a=false;
        b=true;
        ergebnis=a&&b;
        document.write("false and true: " + ergebnis + "<br>");
        a=false;
        b=false;
        ergebnis=a&&b;
        document.write("false and false: " + ergebnis + "<br>");
    </script>
```

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

And-Wertetabelle in php

```
<!-- Das Programm fuer die and-Wertetabelle
  Dateiname: loesung4.3a.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>And-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die And-Wertetabelle
    </p>
<?php
        $a=true;
        $b=true;
        $ergebnis=$a&&$b;
        echo "true and true: $ergebnis <br>";
        $a=true;
        $b=false;
        $ergebnis=$a&&$b;
        echo "true and false: $ergebnis <br>";
        $a=false;
        $b=true;
        $ergebnis=$a&&$b;
        echo "false and true: $ergebnis <br>";
        $a=false;
        $b=false;
        $ergebnis=$a&&$b;
        echo "false and false: $ergebnis <br>";
?>
</body>
</html>
```

Or-Wertetabelle in JavaScript

```
<!-- Programm fuer die or-Wertetabelle
  Dateiname: loesung4.3b.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Or-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die Or-Wertetabelle
    </p>
    <script LANGUAGE = "JavaScript">
        var a
```

```
        var b;
        var ergebnis;
        a=true;
        b=true;
        ergebnis=a||b;
        document.write("true or true: " + ergebnis + "<br>");
        a=true;
        b=false;
        ergebnis=a||b;
        document.write("true or false: " + ergebnis + "<br>");
        a=false;
        b=true;
        ergebnis=a||b;
        document.write("false or true: " + ergebnis + "<br>");
        a=false;
        b=false;
        ergebnis=a||b;
        document.write("false or false: " + ergebnis + "<br>");
    </script>
</body>
</html>
```

Or-Wertetabelle in php

```
<!-- Das Programm fuer die or-Wertetabelle
  Dateiname: loesung4.3b.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Or-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die Or-Wertetabelle
    </p>
<?php
        $a=true;
        $b=true;
        $ergebnis=$a||$b;
        echo "true or true: $ergebnis <br>";
        $a=true;
        $b=false;
        $ergebnis=$a||$b;
        echo "true or false: $ergebnis <br>";
        $a=false;
        $b=true;
        $ergebnis=$a||$b;
        echo "false or true: $ergebnis <br>";
        $a=false;
        $b=false;
        $ergebnis=$a||$b;
        echo "false or false: $ergebnis <br>";
?>
```

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

Not-Wertetabelle in JavaScript

```
<!-- Programm fuer die not-Wertetabelle
  Dateiname: loesung4.3c.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Not-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die Not-Wertetabelle
    </p>
    <script LANGUAGE = "JavaScript">
        var a
        var ergebnis;
        a=true;
        ergebnis=!a;
        document.write("not true: " + ergebnis + "<br>");
        a=false;
        ergebnis=!a;
        document.write("not false: " + ergebnis + "<br>");
    </script>
</body>
</html>
```

Not-Wertetabelle in php

```
<!-- Das Programm fuer die not-Wertetabelle
  Dateiname: loesung4.3c.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Not-Wertetabelle</title>
</head>
<body>
    <p>
        Testprogramm fuer die Not-Wertetabelle
    </p>
<?php
        $a=true;
        $ergebnis=!$a;
        echo "not true: $ergebnis <br>";
        $a=false;
        $ergebnis=!$a;
        echo "not false: $ergebnis <br>";
?>
</body>
</html>
```

Lösung Aufgabe 4.4 In diesen Beispielen dieses Kapitels hatten die Variablen selber keine besondere Bedeutung, sie dienten nur dazu, die Wirkungsweise von Operatoren zu veranschaulichen. Daher macht es auch keinen besonderen Sinn, den Variablen sprechende Namen zu geben; es gibt ja eigentlich auch keine.

Lösungen Kapitel 5

Lösung Aufgabe 5.1

JavaScript

```
<!-- Bank Teil 1
  Dateiname: losung5.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank Teil 1 </title>
</head>
<body>
    <script language = "JavaScript">
        var immobilienpreis;
        var eigenkapital;
        var zinssatz=0.05;
        var tilgung=0.01;
        var kreditbetrag;
        var monatlicheBelastung;
        immobilienpreis=prompt("Bitte geben Sie den Preis der Immobilie ein!","");
        eigenkapital=prompt("Bitte geben Sie das Eigenkapital ein!","");
        immobilienpreis=parseFloat(immobilienpreis);
        eigenkapital=parseFloat(eigenkapital);
        kreditbetrag= immobilienpreis-eigenkapital;
        monatlicheBelastung=(kreditbetrag*(zinssatz+tilgung))/12;
        document.write ("Die monatliche Belastung betr&auml;gt " +
                                    monatlicheBelastung);
    </script>
</body>
</html>
```

php

```
<!-- Bank Teil 1
  Dateiname: loesung5.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank 1</title>
</head>
<body>
<h2>
    Monatliche Belastung berechnen
</h2>
<form name="bank1" action="./loesung5.1.php" method="post">
    <table border>
        <tr>
            <td>
                Preis der Immobilie
            </td>
            <td>
                <input type="text" name="immobilienpreis" size=12>
```

```
            </td>
        </tr>
        <tr>
            <td>
               Eigenkapital
            </td>
            <td>
                <input type="text" name="eigenkapital" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

```
<!-- Programm Bannk 1
  Dateiname: loesung5.1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank 1</title>
</head>
<body>
<?php
    define("ZINSSATZ", "0.05");
    define("TILGUNG", "0.01");
    $kreditbetrag=$immobilienpreis-$eigenkapital;
    $monatlicheBelastung=($kreditbetrag*(ZINSSATZ+TILGUNG))/12;
    echo "Die monatliche Belastung betr&auml;gt $monatlicheBelastung!";
?>
</body>
</html>
```

Lösung Aufgabe 5.2 Die Lösung dieser Aufgabenstellung ist relativ einfach. Wir benutzen Beispiel 5.7. Einziges Problem dort ist, dass die Flugzeit in Sekunden eingegeben wird. Aber das macht uns weiter nichts, wir lesen die Stunden, Minuten und Sekunden der Flugzeit ein und rechnen dann gnadenlos alles in Sekunden um[1].

JavaScript

```
<!-- Raketenbeispiel 1 der Aufgaben
  Dateiname: loesung5.2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
```

[1] Das macht man übrigens, indem man die Stunden mit 3600, die Minuten mit 60 und die Sekunden mit 1 multipliziert :-).

```
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Abflugzeit einer Rakete <br>
    und sodann die Flugzeit ein.<br>
    Die Ankunftszeit wird berechnet.<br>
    <script language = "JavaScript">
    var sekunden;
    var minuten;
    var stunden;
    var flugzeitStunden;
    var flugzeitMinuten;
    var flugzeitSekunden;
    var flugzeit;
    //Einlesen
    stunden=prompt("Bitte geben Sie die Stunden ein","");
    minuten=prompt("Bitte geben Sie die Minuten ein","");
    sekunden=prompt("Bitte geben Sie die Sekunden ein","");
    flugzeitStunden=prompt("Bitte geben Sie die Stunden der " +
                           "Flugzeit ein","");
    flugzeitMinuten=prompt("Bitte geben Sie die Minuten der " +
                           "Flugzeit ein","");
    flugzeitSekunden=prompt("Bitte geben Sie die Sekunden der " +
                           "Flugzeit ein","");

    //Umwandeln
    stunden=parseInt(stunden);
    minuten=parseInt(minuten);
    sekunden=parseInt(sekunden);
    flugzeitStunden=parseInt(flugzeitStunden);
    flugzeitMinuten=parseInt(flugzeitMinuten);
    flugzeitSekunden=parseInt(flugzeitSekunden);
    // flugzeit in sekunden umrechnen
    flugzeit=flugzeitStunden*3600+flugzeitMinuten*60+flugzeitSekunden;
    //Berechnen, zuerst Sekunden und Minuten
    sekunden=sekunden+flugzeit;
    minuten=minuten+Math.floor(sekunden/60);
    sekunden=sekunden%60;
    //nun minuten und stunden
    stunden=stunden+Math.floor(minuten/60);
    minuten=minuten%60;
    //ausgeben
    document.write("Die Ankunftszeit ist: <br>" +
                   stunden + ":" + minuten + ":" +
                   sekunden);
    </script>
</body>
</html>
```

php

```
<!-- raketenaufgabe 1
```

```
    Dies ist die Eingabemaske
  Dateiname: loesung5.2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Abflugzeit einer Rakete <br>
    und sodann die Flugzeit ein.<br>
    Die Ankunftszeit wird berechnet.<br> <hr>
<form name="raketen1" action="./loesung5.2.php" method="post">
    <table border>
        <tr>
            <td>
                Abflugzeit Stunden
            </td>
            <td>
                <input type="text" name="stunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Abflugzeit Minuten
            </td>
            <td>
                <input type="text" name="minuten" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Abflugzeit Sekunden
            </td>
            <td>
                <input type="text" name="sekunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Flugzeit Stunden
            </td>
            <td>
                <input type="text" name="flugzeitStunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Flugzeit Minuten
            </td>
            <td>
                <input type="text" name="flugzeitMinuten" size=12>
            </td>
        </tr>
```

```
        <tr>
            <td>
                Flugzeit Sekunden
            </td>
            <td>
                <input type="text" name="flugzeitSekunden" size=12>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

```
<!-- Raketenbeispiel Teil 1
    Dies ist die Eingabemaske
  Dateiname: loesung5.2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel Teil 1</title>
</head>
<body>
<h2>
    Raketenbeispiel Teil 1: Das Ergebnis
</h2>
<?php
    // umrechnen
    $flugzeit=$flugzeitStunden*3600+$flugzeitMinuten*60+$flugzeitSekunden;
    //Berechnen, zuerst Sekunden und Minuten
    $sekunden=$sekunden+$flugzeit;
    $minuten=$minuten+floor($sekunden/60);
    $sekunden=$sekunden%60;
    //nun minuten und stunden
    $stunden=$stunden+floor($minuten/60);
    $minuten=$minuten%60;
    //ausgeben
    echo("Die Ankunftszeit ist: <br>" .
                    "$stunden:$minuten:$sekunden");
?>
</body>
</html>
```

Lösung Aufgabe 5.3 Diese Aufgabe ist wiederum nicht ganz so einfach. Daher denken wir hier vor der Implementierung etwas nach. Wie bei Beispiel 5.7 ist der grobe Rahmen klar: Wir müssen die Start- und Landezeit einlesen, dann die Flugzeit berechnen und letztlich das Ergebnis ausgeben. Problematisch ist auch hier wieder die Berechnung. Doch auch hier nutzen wir wieder Beispiel 5.7.

Dort haben wir u.a. gelernt, wie man Sekunden in Stunden, Minuten und Sekunden umrechnet. Das geht nämlich folgendermaßen:

- Wir nehmen die gegebenen Sekunden, führen eine Integerdivision durch 60 durch. Wir erhalten die in den Sekunden enthaltenen Minuten.
- Wir rechnen “gegebene Sekunden” modulo 60 und erhalten die wirklichen Sekunden.
- Wir nehmen unsere im ersten Schritt berechneten Minuten und integerdividieren diese nochmal durch 60, weil da ja auch Stunden enthalten sein können. Das Ergebnis dieser Integerdivision sind die tatsächlichen in den ursprünglichen Sekunden enthaltenen Stunden.
- Die tatsächlichen Minuten erhalten wir, indem wir “berechnete Minuten” modulo 60 rechnen.

Nun müssen wir eigentlich nur noch an die Sekunden der Flugzeit kommen. Das ist aber wiederum nicht so schwer. Wir rechnen zunächst die Start- und Landezeit in Sekunden um (wie das geht zeigt die Lösung Aufgabe 5.2). Dann ziehen wir die Startzeit in Sekunden von der Landezeit in Sekunden ab und erhalten die Flugzeit in Sekunden. Diese rechnen wir dann wieder in Stunden, Minuten und Sekunden zurück. Zusammenfassend ergeben sich folgende Schritte:

1. Umrechnen der Startzeit in Sekunden
2. Umrechnen der Landezeit in Sekunden
3. Flugzeit in Sekunden = Landezeit in Sekunden - Startzeit in Sekunden
4. vorläufige Minuten der Flugzeit = Flugzeit in Sekunden integerdividiert durch 60
5. Sekunden der Flugzeit = Flugzeit in Sekunden modulo 60
6. Stunden der Flugzeit = vorläufige Minuten der Flugzeit integerdividiert durch 60
7. Minuten der Flugzeit = vorläufige Minuten der Flugzeit modulo 60

JavaScript

```
<!-- Raketenbeispiel 2 der Aufgaben
  Dateiname: loesung5.3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und sodann die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br>
    <script language = "JavaScript">
        var startSekunden;
        var startMinuten;
        var startStunden;
        var startzeitInSekunden;
        var landeStunden;
        var landeMinuten;
```

```
        var landeSekunden;
        var landezeitInSekunden;
        var flugzeitInSekunden;
        var flugzeitStunden;
        var flugzeitMinuten;
        var flugzeitSekunden;
        //Einlesen
        startStunden=prompt("Bitte geben Sie die Stunden der Startzeit ein","");
        startMinuten=prompt("Bitte geben Sie die Minuten der Startzeit ein","");
        startSekunden=prompt("Bitte geben Sie die Sekunden der Startzeit ein","");
        landeStunden=prompt("Bitte geben Sie die Stunden der " +
                            "Landezeit ein","");
        landeMinuten=prompt("Bitte geben Sie die Minuten der " +
                            "Landezeit ein","");
        landeSekunden=prompt("Bitte geben Sie die Sekunden der " +
                            "Lande ein","");

        //Umwandeln
        startStunden=parseInt(startStunden);
        startMinuten=parseInt(startMinuten);
        startSekunden=parseInt(startSekunden);
        landeStunden=parseInt(landeStunden);
        landeMinuten=parseInt(landeMinuten);
        landeSekunden=parseInt(landeSekunden);
        // start-und landezeit in sekunden umrechnen
        startzeitInSekunden=startStunden*3600+startMinuten*60+startSekunden;
        landezeitInSekunden=landeStunden*3600+landeMinuten*60+landeSekunden;
        // flugzeitInSekunden berechnen
        flugzeitInSekunden=landezeitInSekunden-startzeitInSekunden;
        //Flugzeit umrechnen, zuerst Sekunden und Minuten
        flugzeitMinuten=Math.floor(flugzeitInSekunden/60);
        flugzeitSekunden=flugzeitInSekunden%60;
        //nun minuten und stunden
        flugzeitStunden=Math.floor(flugzeitMinuten/60);
        flugzeitMinuten=flugzeitMinuten%60;
        //ausgeben
        document.write("Die Flugzeit betr&auml;gt: <br>" +
                flugzeitStunden + " Stunden <br>" +
                flugzeitMinuten + " Minuten <br>" +
                flugzeitSekunden + " Sekunden <br>");
    </script>
</body>
</html>
```

php

```
<!-- Raketenaufgabe Teil 2
    Dies ist die Eingabemaske
  Dateiname: loesung5.3.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel </title>
```

```
</head>
<body>
    Bitte geben Sie in die Eingabefenster <br>
    die Startzeit einer Rakete <br>
    und die Landezeit ein.<br>
    Die Flugzeit wird berechnet.<br> <hr>
<form name="raketen2" action="./loesung5.3.php" method="post">
    <table border>
        <tr>
            <td>
                Startzeit Stunden
            </td>
            <td>
                <input type="text" name="startStunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Startzeit Minuten
            </td>
            <td>
                <input type="text" name="startMinuten" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Startzeit Sekunden
            </td>
            <td>
                <input type="text" name="startSekunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Landezeit Stunden
            </td>
            <td>
                <input type="text" name="landeStunden" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Landezeit Minuten
            </td>
            <td>
                <input type="text" name="landeMinuten" size=12>
            </td>
        </tr>
        <tr>
            <td>
                Landezeit Sekunden
            </td>
            <td>
                <input type="text" name="landeSekunden" size=12>
```

```
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="Button1" value="Abschicken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>


<!-- Raketenaufgabe Teil 2
    Die Logik
  Dateiname: loesung5.3.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Raketenbeispiel</title>
</head>
<body>
<h2>
    Raketenbeispiel: Das Ergebnis
</h2>
<?php
    // umrechnen
    $startzeitInSekunden=$startStunden*3600+$startMinuten*60+$startSekunden;
    $landezeitInSekunden=$landeStunden*3600+$landeMinuten*60+$landeSekunden;
    // flugzeitInSekunden berechnen
    $flugzeitInSekunden=$landezeitInSekunden-$startzeitInSekunden;
    //Flugzeit umrechnen, zuerst Sekunden und Minuten
    $flugzeitMinuten=floor($flugzeitInSekunden/60);
    $flugzeitSekunden=$flugzeitInSekunden%60;
    //nun minuten und stunden
    $flugzeitStunden=floor($flugzeitMinuten/60);
    $flugzeitMinuten=$flugzeitMinuten%60;
    //ausgeben
    echo ("Die Flugzeit betr&auml;gt: <br>" .
                    "$flugzeitStunden Stunden <br>" .
                    "$flugzeitMinuten Minuten <br>" .
                    "$flugzeitSekunden Sekunden <br>");

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

Lösungen Kapitel 6

Lösung Aufgabe 6.1

JavaScript

```
<!-- Bank einfach
  Dateiname: loesung6.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank einfach</title>
</head>
<body>
    <script language = "JavaScript">
        var betrag;
        var preis;
        var eigenkapital;
        var eigenkapitalQuote;
        var zins=5;
        var tilgung=1;
        var jahresZahlung;
        var monatsZahlung;
        preis=prompt("Geben Sie hier den Preis Ihrer gewünschten Eigentumswohnung an!
        eigenkapital = prompt("Geben Sie hier Ihr Eigenkapital an!","");
        preis= parseFloat(preis);
        eigenkapital=parseFloat(eigenkapital);
        betrag = preis-eigenkapital;
        eigenkapitalQuote=(eigenkapital/preis)*100;
        if (eigenkapitalQuote<30)
        {
            document.write("<B> Ihre Eigenkapitalquote betr&auml;gt nur " +
                        eigenkapitalQuote + " Prozent!</B></BR>");
            document.write("<B> Wir k&ouml;nnen die Finanzierung " +
                        "leider nicht &uuml;bernehmen!</B>");
        }
        else
        {
            jahresZahlung=(betrag/100)*(zins+tilgung);
            monatsZahlung=jahresZahlung/12;
            document.write("<B> Ihre monatliche Belastung beträgt: " +
                        monatsZahlung + " DM </B>");
        }
    </script>
</body>
</html>
```

php

```
<!-- Programm Bank einfach
  Dateiname: loesung6.1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
```

```
<head>
  <title> Bank einfach Teil 2</title>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='BankEinfach2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Preis der Immobilie
                    </td>
                    <td>
                        <input type="text" name="immobilienpreis" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Eigenkapital
                    </td>
                    <td>
                        <input type="text" name="eigenkapital" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        define("ZINSSATZ", "0.05");
        define("TILGUNG", "0.01");
        $eigenkapitalQuote=($eigenkapital/$immobilienpreis)*100;
        if ($eigenkapitalQuote<30)
        {
            echo("<B> Ihre Eigenkapitalquote betr&auml;gt nur " .
                        "$eigenkapitalQuote Prozent!</B></BR>");
            echo("<B> Wir k&ouml;nnen die Finanzierung leider " .
                        "nicht &uuml;bernehmen!</B>");
        }
        else
        {
            $kreditbetrag=$immobilienpreis-$eigenkapital;
            $monatlicheBelastung=($kreditbetrag*(ZINSSATZ+TILGUNG))/12;
            echo "Die monatliche Belastung betr&auml;gt $monatlicheBelastung!";
```

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

Lösung Aufgabe 6.2 Bei dieser Lösung müssen wir wieder überlegen. Wir werden, bevor wir mit der Programmierung beginnen, den Algorithmus herleiten.

Die grobe Vorgehensweise ist:

1. Immobilienpreis, Eigenkapital und Neubau (J/N) einlesen
2. Wert der Immobilie bestimmen
3. Kreditsumme bestimmen
4. Monatliche Belastung berechnen
5. Monatliche Belastung ausgeben

Bis auf "Monatliche Belastung berechnen" ist alles einfach und mit Standardmethoden lösbar. Bei Monatliche Belastung berechnen sind für die Berechnung der Zinsen vier Fälle denkbar:

1. Die Kreditsumme ist kleiner als 60% des Wertes. Der gesamte Kredit wird mit 6,25% verzinst.
2. Die Kreditsumme liegt zwischen 60% und 80% des Wertes. Der Teil des Kredits, der 60% des Immobilienwerts entspricht, wird mit 6,25% verzinst, der Rest mit 7%.
3. Die Kreditsumme liegt zwischen 80% und 100% des Wertes. Der Teil des Kredits, der 60% des Immobilienwerts entspricht, wird mit 6,25% verzinst, der Teil, der 20% des Immobilienwerts entspricht (dies ist der Bereich zwischen 60% und 80%), mit 7% und der Rest mit 7,5%.
4. Die Kreditsumme liegt über 100% des Wertes. Der Teil des Kredits, der 60% des Immobilienwerts entspricht, wird mit 6,25% verzinst, der erste Teil, der 20% des Immobilienwerts entspricht (dies ist der Bereich zwischen 60% und 80%), mit 7%, der nächste Teil, der 20% des Immobilienwerts entspricht (dies ist der Bereich zwischen 80% und 100%), mit 7,5% und der Rest mit 8,5%.

Zum Abschluss müssen wir noch

- den Tilgungsanteil berechnen (kredit*tilgung) und
- die Jahresbelastung auf den Monat umrechnen (Division durch 12).

Dies formulieren wir jetzt in Pseudocode:

1. if(kredit<wert*0.6)
   {
   (a) jahresbelastung=kredit*0.0625

   }
2. if((kredit>wert*0.6) und (kredit<=wert*0.8))
   {

   (a) jahresbelastung=wert*0.6*0.0625+(kredit-wert*0.6)*0.7

   }

3. if((kredit>wert*0.8) und (kredit<=wert))
   {
   (a) jahresbelastung=wert*0.6*0.0625+wert*0.2*0.7+(kredit-wert*0.8)*0.75

   }

4. if((kredit>wert)
   {
   (a) jahresbelastung=wert*0.6*0.0625+wert*0.2*0.7+wert*0.2*0.75+(kredit-wert)*0.85

   }

5. jahresbelastung=jahresbelastung+kredit*tilgung

6. monatsbelastung=jahresbelastung/12

Bevor wir diesen Pseudocode in ein Programm umsetzen, bedenken wir noch, dass Banken ihre Zinssätze häufig ändern. Ebenso können die Bereiche, für die verschiedene Zinsen gelten, variieren. Aus Kapitel 4.5 wissen wir, was zu tun ist. Wir sollten sowohl Zinsen, als auch die Bereiche, in denen die unterschiedlichen Zinssätze gelten, Konstanten[2] zuweisen. Im Programm benutzen wir dann die Konstanten.

Wenn wir so vorgehen, müssen wir uns jetzt allerdings überlegen, wie wir an die Zahlen (0.6, bzw. 0.2) kommen, mit denen wir den Wert der Immobilie jeweils multiplizieren. Das ist aber auch nicht schwer. Definieren wir als Konstanten (JavaScript-Notation):

```
var grenze1=0.6;
var grenze2=0.8;
var grenze3=1;
```

Dann entspricht:

```
grenze1 der 0.6 des Pseudocodes,
grenze2-grenze1 der ersten 0.2 des Pseudocodes und
grenze3-grenze2 der zweiten 0.2 des Pseudocodes.
```

JavaScript

```
<!-- Bank schwierig
  Dateiname: loesung6.2.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank schwierig</title>
</head>
<body>
    <script language = "JavaScript">
        var kreditbetrag;
        var preis;
        var eigenkapital;
```

[2]Bei JavaScript Variablen, die wir im Programm nicht mehr ändern.

```
var eigenkapitalQuote;
var jahresZahlung;
var monatsZahlung;
var neubau;
var wert;

// Konstantendefinitionen: Die Grenzen der Zinsklassen und die
// Zinsen selbst sollen moeglichst leicht aenderbar sein.
// Da Javascript ja keine Konstanten kennt, muessen wir
// Variablen nehmen, die wir im Programmverlauf aber nicht aendern.
var grenze1=0.6;
var grenze2=0.8;
var grenze3=1;
var prozentBisGrenze1 = 0.06;
var prozentZwischenGrenze1Grenze2 = 0.07;
var prozentZwischenGrenze2Grenze3  = 0.075;
var prozentUeberGrenze3= 0.085;

preis=prompt("Geben Sie hier den Preis Ihrer gewünschten Eigentumswohnung an!
eigenkapital=prompt("Geben Sie hier Ihr Eigenkapital an!","");
tilgung=prompt("Geben Sie hier die Tilgung ein!","");
neubau=prompt("Ist es ein Neubau (J/N)","");
preis= parseFloat(preis);
eigenkapital=parseFloat(eigenkapital);
tilgung=parseFloat(tilgung);
kreditbetrag = preis-eigenkapital;
if((neubau=="J")|| (neubau=="j"))
{
    wert=preis*0.8;
}
else
{
    wert=preis;
}
if(kreditbetrag<=wert*grenze1)
{
    jahresBelastung=kreditbetrag*prozentBisGrenze1;
}
if((kreditbetrag>wert*grenze1) && (kreditbetrag<=wert*grenze2))
{
    jahresBelastung=wert*grenze1*prozentBisGrenze1 +
            (kreditbetrag-wert*grenze1)*prozentZwischenGrenze1Grenze2;
}
if((kreditbetrag>wert*grenze2) && (kreditbetrag<=wert*grenze3))
{
    jahresBelastung=wert*grenze1*prozentBisGrenze1 +
            wert*(grenze2-grenze1)*prozentZwischenGrenze1Grenze2 +
            (kreditbetrag-wert*grenze2)*prozentZwischenGrenze2Grenze3;
}
if(kreditbetrag>wert*grenze3)
{
    jahresBelastung=wert*grenze1*prozentBisGrenze1 +
            wert*(grenze2-grenze1)*prozentZwischenGrenze1Grenze2 +
            wert*(grenze3-grenze2)*prozentZwischenGrenze2Grenze3 +
```

```
                    (kreditbetrag-wert*grenze3)*prozentUeberGrenze3;
        }
        jahresBelastung=jahresBelastung+kreditbetrag*tilgung;
        monatsBelastung=jahresBelastung/12;
        document.write("Die monatliche Belastung betr&auml;gt " +
                        monatsBelastung)
    </script>
</body>
</html>
```

php

```
<!--  Bank schwierig
  Dateiname: loesung6.1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Bank schwierig </title>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='BankEinfach2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Preis der Immobilie
                    </td>
                    <td>
                        <input type="text" name="immobilienpreis" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Eigenkapital
                    </td>
                    <td>
                        <input type="text" name="eigenkapital" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Tilgung
                    </td>
                    <td>
                        <input type="text" name="tilgung" size=12>
                    </td>
                </tr>
```

```
                <tr>
                    <td>
                        Neubau (J/N)
                    </td>
                    <td>
                        <input type="text" name="neubau" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        define(GRENZE1,"0.6");
        define(GRENZE2,"0.8");
        define(GRENZE3,"1");
        define(PROZENT_BIS_GRENZE1,"0.06");
        define(PROZENT_ZWISCHEN_GRENZE1_GRENZE2,"0.07");
        define(PROZENT_ZWISCHEN_GRENZE2_GRENZE3,"0.075");
        define(PROZENT_UEBER_GRENZE3,"0.085");
        $kreditbetrag=$immobilienpreis-$eigenkapital;
        if(($neubau=="J")||($neubau=="j"))
        {
            $wert=$immobilienpreis*0.8;
        }
        else
        {
            $wert=$immobilienpreis;
        }
        if($kreditbetrag<=$wert*GRENZE1)
        {
            $jahresBelastung=$kreditbetrag*PROZENT_BIS_GRENZE1;
        }
        if(($kreditbetrag>$wert*GRENZE1) && ($kreditbetrag<=$wert*GRENZE2))
        {
            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    ($kreditbetrag-$wert*GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2;
        }
        if(($kreditbetrag>$wert*GRENZE2) && ($kreditbetrag<=$wert*GRENZE3))
        {
            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    $wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
                    ($kreditbetrag-$wert*GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3;
        }
        if($kreditbetrag>$wert*GRENZE3)
        {
            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    $wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
```

```
                    $wert*(GRENZE3-GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3 +
                    ($kreditbetrag-$wert*GRENZE3)*PROZENT_UEBER_GRENZE3;
        }
        $jahresBelastung=$jahresBelastung+$kreditbetrag*$tilgung;
        $monatsBelastung=$jahresBelastung/12;
        echo("Die monatliche Belastung betr&auml;gt $monatsBelastung!");
    }
?>
</body>
</html>
```

Lösung Aufgabe 6.3

```
<!--  Taschenrechner
  Dateiname: loesung6.3.html  //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Taschenrechner </title>
</head>
<body>
    <script language = "JavaScript">
        var ersterOperand;
        var zweiterOperand;
        var operator;
        var ergebnis;
        // einlesen
        ersterOperand=prompt("Bitte geben Sie den ersten Operanden ein!","");
        operator=prompt("Bitte geben Sie den Operator ein!","");
        zweiterOperand=prompt("Bitte geben Sie den zweiten Operanden ein!","");
        ersterOperand=parseFloat(ersterOperand);
        zweiterOperand=parseFloat(zweiterOperand);
        if(operator=="+")
        {
            ergebnis=ersterOperand+zweiterOperand;
        }
        else
        {
            // war nicht +, vielleicht -
            if(operator=="-")
            {
                ergebnis=ersterOperand-zweiterOperand;
            }
            else
            {
                // auch nicht - vielleicht *
                if(operator=="*")
                {
                    ergebnis=ersterOperand*zweiterOperand;
                }
                else
                {
```

```
                // auch nicht * vielleicht /
                    if(operator=="/")
                    {
                        //Teilen durch 0 verhindern
                        if(zweiterOperand!=0)
                        {
                            ergebnis=ersterOperand/zweiterOperand;
                        }
                        else
                        {
                            ergebnis="Versuch durch 0 zu teilen!";
                        }
                    }
                    else
                    {
                        // auch nicht /, Fehleingabe
                        ergebnis="Falscher Operator eingegeben!";
                    } //schliesst else zu if /
                }//schliesst else zu if *
            }//schliesst else zu if -
        }//schliesst else zu if +
        document.write("Das Ergebnis ist: "+ ergebnis);
    </script>
</body>
</html>
```

Lösung Aufgabe 6.4

```
<!--  Taschenrechner mit Switch
  Dateiname: loesung6.4.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Taschenrechner </title>
</head>
<body>
    <script language = "JavaScript">
        var ersterOperand;
        var zweiterOperand;
        var operator;
        var ergebnis;
        // einlesen
        ersterOperand=prompt("Bitte geben Sie den ersten Operanden ein!","");
        operator=prompt("Bitte geben Sie den Operator ein!","");
        zweiterOperand=prompt("Bitte geben Sie den zweiten Operanden ein!","");
        ersterOperand=parseFloat(ersterOperand);
        zweiterOperand=parseFloat(zweiterOperand);
        switch (operator)
        {
            case "+":
                ergebnis=ersterOperand+zweiterOperand;
                break;
```

```
            case "-":
                ergebnis=ersterOperand-zweiterOperand;
                break;
            case "*":
                ergebnis=ersterOperand*zweiterOperand;
                break;
            case "/":
                if(operator=="/")
                {
                    //Teilen durch 0 verhindern
                    if(zweiterOperand!=0)
                    {
                        ergebnis=ersterOperand/zweiterOperand;
                    }
                    else
                    {
                        ergebnis="Versuch durch 0 zu teilen!";
                    }
                }
                break;
            default:
                ergebnis="Falscher Operator eingegeben!";
        }
        document.write("Das Ergebnis ist: "+ ergebnis);
    </script>
</body>
</html>
```

Lösungen Kapitel 7

Lösung Aufgabe 7.1

php Zunächst die ausgelagerten Funktionen:

```
<?php
// zinsfunktionen
// Datei:loesung7.1.inc.php
// Verzeichnis: includes
    function berechneEigenkapitalquote($eigenkapital, $preis)
    {
        return(($eigenkapital/$preis)*100);
    }
    function berechneMonatlicheBelastung($eigenkapital,$preis)
    {
        define("ZINSSATZ", "0.05");
        define("TILGUNG", "0.01");
        $kreditbetrag=$preis-$eigenkapital;
        $monatlicheBelastung=($kreditbetrag*(ZINSSATZ+TILGUNG))/12;
        return($monatlicheBelastung);
    }
?>
```

Sodann das Hauptprogramm in der html-Datei.

```
<!-- Programm Bank einfach
  Dateiname: loesung7.1.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Bank einfach Teil 2</title>
<?php
    require_once("./includes/loesung7.1.inc.php");
?>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")
    {
    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='BankEinfach2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Preis der Immobilie
                    </td>
                    <td>
                        <input type="text" name="immobilienpreis" size=12>
                    </td>
                </tr>
```

```
                <tr>
                    <td>
                        Eigenkapital
                    </td>
                    <td>
                        <input type="text" name="eigenkapital" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php
    }
    else
    {
        $eigenkapitalQuote=berechneEigenkapitalquote($eigenkapital,$immobilienpreis);
        if ($eigenkapitalQuote<30)
        {
            echo("<B> Ihre Eigenkapitalquote betr&auml;gt nur " .
                        "$eigenkapitalQuote Prozent!</B></BR>");
            echo("<B> Wir k&ouml;nnen die Finanzierung leider " .
                        "nicht &uuml;bernehmen!</B>");
        }
        else
        {

            $monatlicheBelastung=berechneMonatlicheBelastung($eigenkapital,$immobilie
            echo "Die monatliche Belastung betr&auml;gt $monatlicheBelastung!";
        }
    }
?>
</body>
</html>
```

Hier haben wir durch die Funktionen keine Zeile Code gespart. Im Gegenteil, durch die Deklaration der Funktionen sind Codezeilen hinzugekommen. Aber wir haben, wie in den Beispielen des Scripts, die Realisierung eines Algorithmus in eine Funktion ausgelagert. Jeder, der ein Programm zur Zinsberechnung schreiben möchte und Zugang zu unseren Funktionen hat, kann diese benutzen, ohne sich selbst überlegen zu müssen, wie Zinsberechnung funktioniert[3].

Nun noch die Realisierung in JavaScript:

JavaScript Zunächst die ausgelagerten Funktionen:

```
// zinsfunktionen
// Datei:loesung7.1.js
// Verzeichnis: javascript
```

[3]Obwohl dies hier so einfach ist, dass man in Realität nichts ausgelagert hätte, ist halt nur zum Üben!

```
function berechneEigenkapitalquote(eigenkapital,preis)
{
    return((eigenkapital/preis)*100);
}
function berechneMonatlicheBelastung(eigenkapital,preis)
{
    var ZINSSATZ=0.05;
    var TILGUNG=0.01;
    kreditbetrag=preis-eigenkapital;
    monatlicheBelastung=(kreditbetrag*(ZINSSATZ+TILGUNG))/12;
    return(monatlicheBelastung);
}
```

Sodann das Hauptprogramm in der html-Datei.

```
<!-- Bank einfach
  Dateiname: loesung7.1.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank einfach</title>
  <script language = "JavaScript"
               src="./javascript/loesung7.1.js">

  </script>
</head>
<body>
    <script language = "JavaScript">
        var betrag;
        var preis;
        var eigenkapital;
        var eigenkapitalQuote;
        var jahresZahlung;
        var monatsZahlung;
        preis=prompt("Geben Sie hier den Preis Ihrer gewünschten Eigentumswohnung an!
        eigenkapital = prompt("Geben Sie hier Ihr Eigenkapital an!","");
        preis= parseFloat(preis);
        eigenkapital=parseFloat(eigenkapital);
        eigenkapitalQuote=berechneEigenkapitalquote(eigenkapital,preis);
        if (eigenkapitalQuote<30)
        {
            document.write("<B> Ihre Eigenkapitalquote betr&auml;gt nur " +
                        eigenkapitalQuote + " Prozent!</B></BR>");
            document.write("<B> Wir k&ouml;nnen die Finanzierung " +
                        "leider nicht &uuml;bernehmen!</B>");
        }
        else
        {
            monatsZahlung=berechneMonatlicheBelastung(eigenkapital,preis);
            document.write("<B> Ihre monatliche Belastung beträgt: " +
                        monatsZahlung + " DM </B>");
        }
    </script>
```

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

Lösung Aufgabe 7.2

php Zunächst die ausgelagerten Funktionen:

```
<?php
// zinsfunktionen
// Datei:loesung7.2.inc.php
// Verzeichnis: includes
    function berechneWert($neubau,$immobilienpreis)

        if(($neubau=="J")||($neubau=="j"))

            return($immobilienpreis*0.8);

        else

            return($immobilienpreis);




    function berechneMonatlicheBelastung($eigenkapital,
                          $immobilienpreis,$wert,$tilgung)

        define("GRENZE1","0.6");
        define("GRENZE2","0.8");
        define("GRENZE3","1");
        define("PROZENT_BIS_GRENZE1","0.06");
        define("PROZENT_ZWISCHEN_GRENZE1_GRENZE2","0.07");
        define("PROZENT_ZWISCHEN_GRENZE2_GRENZE3","0.075");
        define("PROZENT_UEBER_GRENZE3","0.085");
        $kreditbetrag=$immobilienpreis-$eigenkapital;
        if($kreditbetrag<=$wert*GRENZE1)

            $jahresBelastung=$kreditbetrag*PROZENT_BIS_GRENZE1;

        if(($kreditbetrag>$wert*GRENZE1) && ($kreditbetrag<=$wert*GRENZE2))

            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    ($kreditbetrag-$wert*GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2;

        if(($kreditbetrag>$wert*GRENZE2) && ($kreditbetrag<=$wert*GRENZE3))

            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    $wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
                    ($kreditbetrag-$wert*GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3;
```

```
        if($kreditbetrag>$wert*GRENZE3)

            $jahresBelastung=$wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    $wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
                    $wert*(GRENZE3-GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3 +
                    ($kreditbetrag-$wert*GRENZE3)*PROZENT_UEBER_GRENZE3;

        $jahresBelastung=$jahresBelastung+$kreditbetrag*$tilgung;
        $monatsBelastung=$jahresBelastung/12;
        return($monatsBelastung);

?>
```

Sodann das Hauptprogramm in der html-Datei.

```
<!--  Bank schwierig
  Dateiname: loesung7.2.php //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title> Bank schwierig </title>
<?php
    require_once("./includes/loesung7.2.inc.php");
?>
</head>
<body>
<?php
    // Wir pruefen zuerst ob die Anfrage ueber get oder post erfolgte
    if($REQUEST_METHOD!="POST")

    // erster Aufruf, das Formular muss praesentiert werden
        echo "<form name='BankEinfach2' action='$PHP_SELF' method='post'>";
?>
            <table border>
                <tr>
                    <td>
                        Preis der Immobilie
                    </td>
                    <td>
                        <input type="text" name="immobilienpreis" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Eigenkapital
                    </td>
                    <td>
                        <input type="text" name="eigenkapital" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Tilgung
```

```
                    </td>
                    <td>
                        <input type="text" name="tilgung" size=12>
                    </td>
                </tr>
                <tr>
                    <td>
                        Neubau (J/N)
                    </td>
                    <td>
                        <input type="text" name="neubau" size=12>
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" name="Button1" value="Abschicken">
                    </td>
                </tr>
            </table>
        </form>
<?php

    else

        $wert=berechneWert($neubau,$immobilienpreis);
        $monatsBelastung=berechneMonatlicheBelastung($eigenkapital,
                                        $immobilienpreis,$wert,$tilgung);
        echo("Die monatliche Belastung betr&auml;gt $monatsBelastung!");

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

Auch hier wurde nicht unbedingt Schreibarbeit gespart. Dennoch ist hier die Auslagerung der Berechnungsfunktion schon sinniger. Denken sie an den Fall, dass die Bank ihren Kunden diese Lösung über das Internet anbieten will. Gleichzeitig stellt die Bank aber den eigenen Mitarbeitern ein komfortableres Programm über das eigene Intranet zur Verfügung. Die Berechnung der monatlichen Belastung bleibt aber in beiden Fällen gleich. Beide Programme können also die gleiche Lösung nutzen.

Nun noch die Lösung in JavaScript:

JavaScript Zunächst die ausgelagerten Funktionen:

```
// zinsfunktionen
// Datei:loesung7.2.js
// Verzeichnis: javascript
    function berechneWert(neubau,immobilienpreis)
    {
        if((neubau=="J")||(neubau=="j"))
        {
            return(immobilienpreis*0.8);
        }
        else
        {
```

```
            return(immobilienpreis);
        }

    }


    function berechneMonatlicheBelastung(eigenkapital,
                            immobilienpreis,wert,tilgung)
    {
        // Konstantendefinitionen: Die Grenzen der Zinsklassen und die
        // Zinsen selbst sollen moeglichst leicht aenderbar sein.
        // Da Javascript ja keine Konstanten kennt, muessen wir
        // Variablen nehmen, die wir im Programmverlauf aber nicht aendern.
        var GRENZE1=0.6;
        var GRENZE2=0.8;
        var GRENZE3=1;
        var PROZENT_BIS_GRENZE1=0.06;
        var PROZENT_ZWISCHEN_GRENZE1_GRENZE2=0.07;
        var PROZENT_ZWISCHEN_GRENZE2_GRENZE3=0.075;
        var PROZENT_UEBER_GRENZE3=0.085;

        kreditbetrag=immobilienpreis-eigenkapital;
        if(kreditbetrag<=wert*GRENZE1)
        {
            jahresBelastung=kreditbetrag*PROZENT_BIS_GRENZE1;
        }
        if((kreditbetrag>wert*GRENZE1) && (kreditbetrag<=wert*GRENZE2))
        {
            jahresBelastung=wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    (kreditbetrag-wert*GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2;
        }
        if((kreditbetrag>wert*GRENZE2) && (kreditbetrag<=wert*GRENZE3))
        {
            jahresBelastung=wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
                    (kreditbetrag-wert*GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3;
        }
        if(kreditbetrag>wert*GRENZE3)
        {
            jahresBelastung=wert*GRENZE1*PROZENT_BIS_GRENZE1 +
                    wert*(GRENZE2-GRENZE1)*PROZENT_ZWISCHEN_GRENZE1_GRENZE2 +
                    wert*(GRENZE3-GRENZE2)*PROZENT_ZWISCHEN_GRENZE2_GRENZE3 +
                    (kreditbetrag-wert*GRENZE3)*PROZENT_UEBER_GRENZE3;
        }
        jahresBelastung=jahresBelastung+kreditbetrag*tilgung;
        monatsBelastung=jahresBelastung/12;
        return(monatsBelastung);
    }
```

Sodann das Hauptprogramm in der html-Datei.

```
<!-- Bank schwierig
  Dateiname: loesung7.2.html //-->
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank schwierig</title>
  <script language = "JavaScript"
               src="./javascript/loesung7.2.js">
  </script>
</head>
<body>
    <script language = "JavaScript">
        var kreditbetrag;
        var preis;
        var eigenkapital;
        var eigenkapitalQuote;
        var jahresZahlung;
        var monatsZahlung;
        var neubau;
        var wert;
        preis=prompt("Geben Sie hier den Preis Ihrer gewünschten Eigentumswohnung an!
        eigenkapital=prompt("Geben Sie hier Ihr Eigenkapital an!","");
        tilgung=prompt("Geben Sie hier die Tilgung ein!","");
        neubau=prompt("Ist es ein Neubau (J/N)","");
        preis= parseFloat(preis);
        eigenkapital=parseFloat(eigenkapital);
        tilgung=parseFloat(tilgung);
        wert=berechneWert(neubau,preis);
        monatsBelastung=berechneMonatlicheBelastung(eigenkapital,
                            preis,wert,tilgung);
        document.write("Die monatliche Belastung betr&auml;gt " +
                            monatsBelastung)
    </script>
</body>
</html>
```

Lösungen Kapitel 9

Lösung Aufgabe 9.1 Hier werde ich nicht die vollständige Lösung darstellen. In Beispiel 9.16 müssen die Zeilen

```
    if(istKeineZahl(document.bestellung.plz,
        "PLZ ist keine Zahl!"))
    {
        return false;
    }
```

ersetzt werden durch

```
    if(istKeinePLZ(document.bestellung.plz,
        "PLZ stimmt nicht!"))
    {
        return false;
    }
```

In die Datei feldkontrolle.js müssen folgende Funktionen aufgenommen werden:

```
// PLZ's
// Datei:loesung9.2.1.js
// Verzeichnis: javascript

function anzahlZeichen (feld, anzahl)
{
    return(feld.length == anzahl);
}

function zahlInBereich (zahl, untererGrenze, obereGrenze)
{
    return ((zahl >= obereGrenze) && (zahl <= untererGrenze))
}

function istKeinePLZ (feld, fehlermeldung)
{
    if (istKeineZahl (feld, fehlermeldung))
    {
        return true;
    }
    if (!anzahlZeichen (feld.value, 5)) //PLZ's sind fuenfstellig
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    if (!zahlInBereich (parseFloat(feld.value), 1000, 99999))
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
```

Lösung Aufgabe 9.2

JavaScript Zunächst die Datei mit der Prüffunktion. Hier gibt es nur zwei, eine überprüft, ob der Inhalt eines Input-Feldes eine positive Zahl ist, die zweite, ob die erste übergebene Zahl größer als die zweite ist:

```
// zinsfunktionen
// Datei:loesung9.2.1.js
// Verzeichnis: javascript
function istKeinePositiveZahl(feld, fehlermeldung)
{
    if(isNaN(feld.value)||(feld.value)<=0)
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
function istGroesser(zahl1, zahl2, fehlermeldung)
{
    if(zahl1>zahl2)
    {
        alert(fehlermeldung);
        return true;
    }
    return false;
}
```

Als zweites die Berechnungsfunktionen, die kennen Sie ja bereits. Der einzige Unterschied ist, dass der Funktion berechneMonatlicheBelastung auch der Zinssatz und die Tilgung mit übergeben wird:

```
// zinsfunktionen
// Datei:loesung9.2.2.js
// Verzeichnis: javascript
function berechneEigenkapitalquote(eigenkapital, preis)
{
    return((eigenkapital/preis)*100);
}
function berechneMonatlicheBelastung(eigenkapital,preis, zins, tilgung)
{
    kreditbetrag=preis-eigenkapital;
    monatlicheBelastung=(kreditbetrag*((zins+tilgung)/100))/12;
    return(monatlicheBelastung);
}
```

Und zum Schluss die html-Datei:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank einfach JavaScript</title>
    <script language = "JavaScript"
                src="./javascript/loesung9.2.1.js">

   </script>
   <script language = "JavaScript"
                src="./javascript/loesung9.2.2.js">

  </script>
  <script language="JavaScript">
    function rechne()
    {
        if(istKeinePositiveZahl(document.bank.preis, "Preis nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.eigenkapital, "Eigenkapital nicht posit
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.zins, "Zins nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.tilgung, "Tilgung nicht positiv"))
        {
            return false;
        }
        preis=parseFloat(document.bank.preis.value);
        eigenkapital=parseFloat(document.bank.eigenkapital.value);
        zins=parseFloat(document.bank.zins.value);
        tilgung=parseFloat(document.bank.tilgung.value);
        if (istGroesser(eigenkapital, preis,
                    "Eigenkapital groesser Preis"))
        {
            return false;
        }
        if(zins < 5)
        {
            alert ("Zinnssatz niediger als unserer!")
        }
        if(tilgung > 4)
        {
            alert ("Tilgung unerwartet hoch!")
        }
        eigenkapitalQuote=berechneEigenkapitalquote(eigenkapital,preis);
        if (eigenkapitalQuote<30)
        {
            alert("Ihre Eigenkapitalquote ist zu niedrig!");
            return false;
        }
```

```
        document.bank.monatlicheBelastung.value=
                            berechneMonatlicheBelastung
                            (eigenkapital,preis, zins, tilgung);

    }
  </script>
</head>
<body>
  <form name="bank">
  <table border="1">
  <tr>
    <td> Preis der Immobilie </td>
    <td> <input type="text" name="preis" size=12> </td>
  </tr>
  <tr>
    <td> Eigenkapital </td>
    <td> <input type="text" name="eigenkapital" size=12> </td>
  </tr>
  <tr>
    <td> Zins </td>
    <td> <input type="text" name="zins" size=12> </td>
  </tr>
  <tr>
    <td> Tilgung </td>
    <td> <input type="text" name="tilgung" size=12> </td>
  </tr>
  <tr>
    <td> Monatliche Belastung </td>
    <td> <input type="text" name="monatlicheBelastung" size=12> </td>
  </tr>
  <tr>
    <td> <input type="button" onclick="rechne()" value="Rechnen"></td>
    <td> <input type="reset" onclick="return(confirm('Alles löschen'))"> </td>
  </tr>
  </table>
  </form>
</body>
</html>
```

php Die JavaScript-Prüffunktionen bleiben die gleichen. Daher verzichte ich auf eine erneute Darstellung. Die Berechnungsfunktionen sind eigentlich auch aus den Lösungen zu Kapitel 7 bekannt. Einziger Unterschied ist, dass, wie ja auch bei der JavaScript-Lösung, Zins und Tilgung mit übergeben werden:

```
<?php
// zinsfunktionen
// Datei:loesung9.2.inc.php
// Verzeichnis: includes
    function berechneEigenkapitalquote($eigenkapital, $preis)
    {
        return(($eigenkapital/$preis)*100);
    }
```

```
    function berechneMonatlicheBelastung($eigenkapital,$preis,$zins,$tilgung)
    {
        $kreditbetrag=$preis-$eigenkapital;
        $monatlicheBelastung=($kreditbetrag*(($zins+$tilgung)/100))/12;
        return($monatlicheBelastung);
    }
?>
```

Die php-datei unterscheidet sich jetzt auch nicht sehr von der weiter oben dargestellten JavaScript-Lösung. Das Ausgabefeld im Formular verschwindet. Schließlich können wir von php aus nicht in eine bereits auf dem Client dargestellte Seite schreiben. Dafür kontrollieren wir, wie immer bei php, mit welcher “Request-Method” die Datei aufgerufen wurde. Ist dies nicht “post”, blenden wir das Formular mit den Eingabefeldern auf. Um Übertragungen der Seite bei ausgeschaltetem JavaScript zu vermeiden, wird die Eingabenkontrollfunktion über den onClick-Event-Handler eines normalen html-Button mit dem Formular verbunden. Abgeschickt wird dann vermittels der submit()-Methode des JavaScript-Formular-Objekts.

Die Berechnungen erfolgen dann serverseitig mit php und das Ergebnis wird dargestellt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
<?php
    require_once("./includes/loesung9.2.inc.php");
?>
  <title>Bank einfach php und javaScript</title>
    <script language = "JavaScript"
                src="./javascript/loesung9.2.1.js">

   </script>
  <script language="JavaScript">
    function rechne()
    {
        if(istKeinePositiveZahl(document.bank.preis, "Preis nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.eigenkapital, "Eigenkapital nicht posit
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.zins, "Zins nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.tilgung, "Tilgung nicht positiv"))
        {
            return false;
        }
        preis=parseFloat(document.bank.preis.value);
        eigenkapital=parseFloat(document.bank.eigenkapital.value);
        zins=parseFloat(document.bank.zins.value);
        tilgung=parseFloat(document.bank.tilgung.value);
```

```
        if (istGroesser(eigenkapital, preis,
                    "Eigenkapital groesser Preis"))
        {
            return false;
        }
        if(zins < 5)
        {
            alert ("Zinnssatz niediger als unserer!")
        }
        if(tilgung > 4)
        {
            alert ("Tilgung unerwartet hoch!")
        }
        document.bank.action=document.bank.anWen.value;
        document.bank.method="post";
        document.bank.submit();
    }
  </script>
</head>
<body>
<?php
    if($REQUEST_METHOD!="POST")
    // erster Aufruf: Formular darstellen!
    {
?>
  <form name="bank">
  <input type="hidden" name="anWen" value="<?php echo $PHP_SELF ?>">
  <table border="1">
  <tr>
    <td> Preis der Immobilie </td>
    <td> <input type="text" name="preis" size=12> </td>
  </tr>
  <tr>
    <td> Eigenkapital </td>
    <td> <input type="text" name="eigenkapital" size=12> </td>
  </tr>
  <tr>
    <td> Zins </td>
    <td> <input type="text" name="zins" size=12> </td>
  </tr>
  <tr>
    <td> Tilgung </td>
    <td> <input type="text" name="tilgung" size=12> </td>
  </tr>
  <tr>
    <td> <input type="button" onclick="rechne()" value="Rechnen"></td>
    <td> <input type="reset" onclick="return(confirm('Alles löschen'))"> </td>
  </tr>
  </table>
  </form>
<?php
    }
    else
    {
```

```
        $eigenkapitalQuote=berechneEigenkapitalquote($eigenkapital,$preis);
        if ($eigenkapitalQuote<30)
        {
            echo "Ihre Eigenkapitalquote ist zu niedrig!";
            exit();
        }
        $monatlicheBelastung=
                            berechneMonatlicheBelastung
                            ($eigenkapital, $preis, $zins, $tilgung);
        echo "Ihre monatliche Belastung betr&auml;gt: $monatlicheBelastung";
    }
?>
</body>
</html>
```

Lösung Aufgabe 9.3

JavaScript Verblüffenderweise ist die Lösung dieser Aufgabe einfacher als die Lösung der vorhergehenden Aufgabe. Denn das wirklich schwierige hier war die Berechnung der monatlichen Belastung unter Berücksichtigung der verschiedenen Zinsklassen und des Wertes des Hauses. Das haben wir aber bereits programmiert und in eine Funktion ausgelagert (vgl. Lösung 7.2). Diese Funktion braucht nun überhaupt nicht geändert werden. Wir binden einfach die “include”-Datei aus Lösung 7.2 ein und rufen die beiden dort programmierten Funktionen auf.

Auch die Datei mit der Prüffunktionen ändert sich nur unwesentlich. Zu den zwei bereits realisierten Funktionen, die überprüfen, ob der Inhalt eines Input-Feldes eine positive Zahl ist und ob die erste übergebene Zahl größer als die zweite ist, kommt eine Prüfroutine, in der wir testen, ob der Wert eines Input-Feldes “j” oder “n” ist:

```
// zinsfunktionen
// Datei:loesung9.2.1.js
// Verzeichnis: javascript
function istKeinePositiveZahl(feld, fehlermeldung)
{
    if(isNaN(feld.value)||(feld.value)<=0)
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
function istGroesser(zahl1, zahl2, fehlermeldung)
{
    if(zahl1>zahl2)
    {
        alert(fehlermeldung);
        return true;
    }
    return false;
```

```
}
function istWederJNochN(feld, fehlermeldung)
{
    if((feld.value!="j")&&(feld.value!="J")&&
            (feld.value!="n")&&(feld.value!="N"))
    {
        feld.focus();
        alert(fehlermeldung);
        return true;
    }
    return false;
}
```

Die Berechnungsfunktionen stelle ich nicht noch einmal da, sie ändern sich ja nicht (vgl. Lösung 7.2). Fehlt nur noch die html-Datei:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Bank einfach JavaScript</title>
    <script language = "JavaScript"
                src="./javascript/loesung9.3.1.js">

   </script>
   <script language = "JavaScript"
                src="./javascript/loesung7.2.js">

  </script>
  <script language="JavaScript">
    function rechne()
    {
        if(istKeinePositiveZahl(document.bank.preis, "Preis nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.eigenkapital, "Eigenkapital nicht posit
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.tilgung, "Tilgung nicht positiv"))
        {
            return false;
        }
        if(istWederJNochN(document.bank.neubau,
            "In dieses Feld muss ’j’ oder ’n’ eingegeben werden!"))
        {
            return false;
        }
        var preis;
        var eigenkapital;
        var tilgung;
        var neubau;
```

```
        var wert;
        preis=parseFloat(document.bank.preis.value);
        eigenkapital=parseFloat(document.bank.eigenkapital.value);
        tilgung=parseFloat(document.bank.tilgung.value);
        neubau=document.bank.neubau.value;
        if (istGroesser(eigenkapital, preis,
                    "Eigenkapital groesser Preis"))
        {
            return false;
        }
        if(tilgung > 4)
        {
            alert ("Tilgung unerwartet hoch!")
        }
        wert=berechneWert(neubau,preis);
        document.bank.monatlicheBelastung.value=
                            berechneMonatlicheBelastung
                            (eigenkapital,preis,wert,tilgung);

    }
  </script>
</head>
<body>
  <form name="bank">
  <table border="1">
  <tr>
    <td> Preis der Immobilie </td>
    <td> <input type="text" name="preis" size=12> </td>
  </tr>
  <tr>
    <td> Eigenkapital </td>
    <td> <input type="text" name="eigenkapital" size=12> </td>
  </tr>
  <tr>
    <td> Tilgung </td>
    <td> <input type="text" name="tilgung" size=12> </td>
  </tr>
  <tr>
    <td> Neubau (j/n) </td>
    <td> <input type="text" name="neubau" size=12> </td>
  </tr>
  <tr>
    <td> Monatliche Belastung </td>
    <td> <input type="text" name="monatlicheBelastung" size=12> </td>
  </tr>
  <tr>
    <td> <input type="button" onclick="rechne()" value="Rechnen"></td>
    <td> <input type="reset" onclick="return(confirm('Alles löschen'))"> </td>
  </tr>
  </table>
  </form>
</body>
</html>
```

php Zur php-Lösung lässt sich grundsätzlich das selbe sagen. Alle Arbeit ist eigentlich schon getan. Daher nun direkt die php-Datei der Lösung:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
<?php
    require_once("./includes/loesung9.2.inc.php");
?>
  <title>Bank einfach php und javaScript</title>
    <script language = "JavaScript"
                src="./javascript/loesung9.2.1.js">

   </script>
  <script language="JavaScript">
    function rechne()
    {
        if(istKeinePositiveZahl(document.bank.preis, "Preis nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.eigenkapital, "Eigenkapital nicht posit
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.zins, "Zins nicht positiv"))
        {
            return false;
        }
        if(istKeinePositiveZahl(document.bank.tilgung, "Tilgung nicht positiv"))
        {
            return false;
        }
        preis=parseFloat(document.bank.preis.value);
        eigenkapital=parseFloat(document.bank.eigenkapital.value);
        zins=parseFloat(document.bank.zins.value);
        tilgung=parseFloat(document.bank.tilgung.value);
        if (istGroesser(eigenkapital, preis,
                    "Eigenkapital groesser Preis"))
        {
            return false;
        }
        if(zins < 5)
        {
            alert ("Zinnssatz niediger als unserer!")
        }
        if(tilgung > 4)
        {
            alert ("Tilgung unerwartet hoch!")
        }
        document.bank.action=document.bank.anWen.value;
        document.bank.method="post";
```

```
        document.bank.submit();
    }
  </script>
</head>
<body>
<?php
    if($REQUEST_METHOD!="POST")
    // erster Aufruf: Formular darstellen!
    {
?>
  <form name="bank">
  <input type="hidden" name="anWen" value="<?php echo $PHP_SELF ?>">
  <table border="1">
  <tr>
    <td> Preis der Immobilie </td>
    <td> <input type="text" name="preis" size=12> </td>
  </tr>
  <tr>
    <td> Eigenkapital </td>
    <td> <input type="text" name="eigenkapital" size=12> </td>
  </tr>
  <tr>
    <td> Zins </td>
    <td> <input type="text" name="zins" size=12> </td>
  </tr>
  <tr>
    <td> Tilgung </td>
    <td> <input type="text" name="tilgung" size=12> </td>
  </tr>
  <tr>
    <td> <input type="button" onclick="rechne()" value="Rechnen"></td>
    <td> <input type="reset" onclick="return(confirm('Alles löschen'))"> </td>
  </tr>
  </table>
  </form>
<?php
    }
    else
    {
        $eigenkapitalQuote=berechneEigenkapitalquote($eigenkapital,$preis);
        if ($eigenkapitalQuote<30)
        {
            echo "Ihre Eigenkapitalquote ist zu niedrig!";
            exit();
        }
        $monatlicheBelastung=
                            berechneMonatlicheBelastung
                            ($eigenkapital, $preis, $zins, $tilgung);
        echo "Ihre monatliche Belastung betr&auml;gt: $monatlicheBelastung";
    }
?>
</body>
</html>
```