# Tutorial

Im Fach Wirtschaftsinformatik ist das Programmieren von Software nicht im Lehrplan vorgeschrieben. Viele Kolleginnen und Kollegen, aber vor allem viele Schülerinnen und Schüler haben aber ein großes Interesse daran, wie die im Unterricht erarbeiteten Modelle in der Praxis funktionieren können. Aus diesem Grund soll in diesem Tutorial eine kleine Anleitung gegeben werden, wie ein Online-Shop programmiert werden kann.

Lehrkräfte im Fach Wirtschaft/Recht haben keine Ausbildung in Programmierung. Es versteht sich daher von selbst, dass ein solches Seminar nur von Lehrkräften angeboten werden kann, die ein eigenes großes Interesse an diesem Lerninhalt haben, verbunden mit dem Willen, sich in diesen auch einzuarbeiten.

In dieser Anleitung werden die wichtigsten Befehle des Online-Shops kurz erklärt. Ziel ist es, den Lehrkräften sowie den Schülerinnen und Schülern, die wenig Erfahrung mit dem Programmieren haben, eine Hilfestellung zu geben. Es werden im Kapitel 3 nacheinander alle Dokumente des Online-Shops einzeln besprochen.

Grundlegende Kenntnisse der Programmierung mit der Programmiersprache HTML und PHP werden in den ersten beiden Kapiteln ausführlich vorangestellt. Diese beiden Kapitel eignen sich daher auch zur Weitergabe an die Schülerinnen und Schüler.

Es ist sicherlich möglich, den Online-Shop sehr viel einfacher oder sehr viel umfangreicher zu gestalten, als es hier dargestellt ist. Nachdem das Seminar über einen längeren Zeitraum besteht, soll das Ergebnis aber nicht zu einfach sein. Die Programmieranfänger im Seminar sollen aber auch nicht überfordert werden. Erweiterungen des Online-Shops sind sicherlich jederzeit möglich.

Ein wichtiger Hinweis am Ende dieser Vorbemerkung: Das Erstellen des Online-Shops macht nur einen Teil des Seminars aus. V.a. die Analyse der Geschäftsprozesse, die Entscheidungsfindung bei der Unternehmens-Gründung, die Kontakte zu den Partnerunternehmen usw. dürfen keinesfalls vernachlässigt werden.

# Inhaltsverzeichnis

# 1. Einführung in HTML

## 1.1 Die Grundstruktur einer HTML-Seite

Zum Erstellen einer Homepage benötigen wir lediglich einen Editor. Zum Ansehen der fertigen Seite (Site) benötigen wir wir natürlich auch einen Browser, z. B. Firefox, Chrome oder den Internet Explorer von Microsoft.

Anhand der Endung eines Dokumentes erkennt man i.d.R. den Dokument-Typ. Das ist bei Homepage-Seiten nicht anders. Hier haben die Dokumente die Endung .htm oder .html, z. B. Seite1.html. Beim Abspeichern von Texten mit Editoren ist darauf zu achten, dass nicht automatisch eine Endung .txt an das Dokument angehängt wird.

Da HTML-Dokumente von überall auf der Welt angesehen werden können verwenden wir weder im Text noch im Dateinamen Umlaute und keine Sonderzeichen. Die Dateinamen erhalten keine Leerzeichen.

HTML-Dokumente besitzen Steuerzeichen, sog. Tags. Diese erkennt man an den eckigen Klammern. Tags sind im Text nicht sichtbar, sondern dienen meist der Formatierung des Textes.

Der Browser erkennt ein HTML-Dokument z. B. am HTML-Tag: `<html>`. Mit diesem Steuerzeichen beginnt jedes HTML-Dokument. Es endet mit dem Tag: `</html>`. Der Schrägstrich zeigt das Ende eines Steuerzeichens an! Die meisten Steuerzeichen enden mit einem eigenen Tag.

Jedes HTML-Formular hat folgenden Aufbau: Es besteht aus Head und Body:

```
<html>
<head>   </head>

<body> <p> Hallo Welt! </p> </body>
</html>
```

Der Text befindet sich im Body. Das `<p>`-Tag bezeichnet einen Absatz des Textes. Dieses Dokument zeigt den Text "Hallo Welt!" an.

Innerhalb des Bodys können nun verschiedenste Tags verwendet werden, um den Text zu formatieren.

Die wichtigsten sind:

| | |
|---|---|
| `<p> </p>` | Absatz |
| `<br>` | neue Zeile (HTML kennt sonst keinen Zeilenumbruch!) |
| `<b> </b>` | Fettdruck |
| `</i></i>` | Kursiv |
| `<h1> </h1>` | Überschriften |
| `<h2> </h2>` | verschiedener Größe |
| `<hr>` | horizontale Linie |
| `<font size="2" face="Arial"> </font>` | Schriftgröße und Art |
| `<p align="center"><p align="right"><p align="left">` | Absatzausrichtung |

## 1.2 Links und Bilder

HTML zeichnet sich v.a. auch durch seine Links aus, mit der eine andere Seite aufgerufen werden kann:

```
<a href="http://www.NaechsteSeite.html">Das Erscheint als Text</a>
```

Innerhalb des <A>-Tags wird das Verweisziel mit `href="http://..."` angegeben. Der Text vor dem Schießenden </A>-Tag gibt an, welchen Text der Verweis hat.

Es lassen sich auch leicht Bilder einfügen. Als Bilder sind jpg-, gif bzw. png-Bilder. Neuerdings werden mit dem svg-Standard auch Vektorgrafiken unterstützt. Der Bildname kann als relative Adresse angegeben werden, z. B. `src="bild.jpg"`. In diesem Fall muss das Bild sich dann im selben Verzeichnis befinden wie die HTML-Seite. Es kann aber auch ein Link aus dem Internet angegeben werden, z. B. `src="http://www...."`.

```
<img src="bild.jpg">
```

Auch die im Dokument gewünschte Größe kann angegeben werden:

```
<img src="bild.jpg" width=200 height=150>
```

Mit dem Attribut `Alt="Irdendein Text"` wird ein alternativer Text angegeben, der angezeigt werden soll, wenn der Browser keine Grafiken darstellen kann oder soll.

```
<img src="bild.jpg" width=200 height=150 alt="Aufkleber - gross">
```

## 1.3 Tabellen

Ein besonders wichtiges Gestaltungsmittel ist die Verwendung von Tabellen. Eine Tabelle besteht aus Zeilen und Zellen. Die Zellen müssen nicht umrahmt werden, damit lassen sich Teile des Dokumentes gut an einer bestimmten Stelle platzieren.

```
<table>
<tr><td>Zelle1</td><td>Zelle2</td></tr>
<tr><td>Zelle3</td><td>Zelle4</td></tr>
<tr><td>Zelle5</td><td>Zelle6</td></tr>
</table>
```

Dabei bedeutet `<tr>` der Beginn einer Zeile und `<td>` der Beginn einer Zelle! Der Rest erklärt sich von selbst.

Die Größe der Tabelle lässt sich ebenfalls steuern:

```
<table width="800" align="center">
<tr><td width="600">Zelle1</td><td width="200">Zelle2</td></tr>
<tr><td>Zelle3</td><td>Zelle4</td></tr>
<tr><td>Zelle5</td><td>Zelle6</td></tr>
</table>
```

In den einzelnen Tags können aber auch Werte für die Farbe, die Schriftart, die Ausrichtung und die Größe der Zelle angegeben werden. %-Angaben bei der Größe einer Zelle beziehen sich auf die Größe des Fensters und variieren daher, wenn der Anwender die Größe des Browserfensters verändert.

```
<table width=500 border=0>
<tr><td width=66%>Zelle1</td><td width=33%>Zelle2</td></tr>
<tr><td>Zelle3</td><td>Zelle4</td></tr>
<tr><td>Zelle5</td><td>Zelle6</td></tr>
</table>
```

## 1.4 Formulare

Oft benötigen wir Eingaben von Benutzern. Eingabefelder werden in HTML über Formulare zur Verfügung gestellt. Ein HTML-Dokument kann die eingegebenen Daten nicht weiterverarbeiten, dafür benötigen wir eine Programmiersprache, z. B. PHP. Die Verarbeitung der Formular-Daten wird in Kapitel 3 behandelt. Aber die Eingabe von Daten und das Übersenden an eine andere Seite (z. B. auch auf einem entfernten Rechner) sind kein großes Problem.

Wir stellen in einem Formular Eingabefelder (`type="text"`) und mindestens einen Absende-Button (`type="submit"`) zur Verfügung. Daneben benötigen wir noch eine Zieladresse, das kann jede beliebige Internet-Adresse sein, z. B.

```
"http://www.irgendwo.com/antwort.html",
```

ide mit dem Attribut `"action=..."` angegeben wird.

Das Formular für zwei Eingabefelder sieht dann so aus:

```
<form action="http://iregendwo.com/antwort.html" methode="GET">
<input type="text" name="Feld1">
<input type="text" name="Feld2">
<input type="submit">
</form>
```

In jedem Formular ist eine Methode "GET" oder "POST" angegeben. POST ist neuer, bei GET werden die zu übermittelnden Wert in der Adresszeile des Browsers angezeigt!

Jedem Eingabefeld geben wir einen Namen (`name="..."`), damit der Empfänger weiß, welche Daten welchen Feldern zugeordnet waren.

Neben den Textfeldern und dem Absende-Button werden im E-Shop Reset-Buttons, mit welchen man die Eingabefelder löschen kann (`<input type="reset">`) und versteckte Textfelder (`<input type="hidden">`) verwendet. Eine Vorbelegung des Textes erfolgt mit dam Attribut "value".

```
<input type="hidden" name="summe" value="100,00">
```

Die Größe eines Eingabefeldes kann über das Attribut `size` angegeben werden.

```
<input type="text" name="summe" size="35">
```

Hinweis:
Neben den hier beschriebenen gibt es eine Vielzahl an weiteren Formularfeldern, z. B. Memo-Felder, Auswahlfelder usw.

## 1.5 Farben in HTML

Farben können mit Wörtern angegeben, z. B. im Body-Tag, um die Hintergrundfarbe zu verändern:

```
<body bgcolor="red">,
```

oder im Font-Tag für die Textfarbe:

```
<font color="red"> rot </font>
```

Die Farben können aber auch in hexadezimaler Schreibweise angegeben werden durch einen sechsstelligen Farbcode von

`<font color="#000000">` für schwarz, bis

`<font color="#FFFFFF">` für weiß.

Immer zwei Ziffern stehen für je einen RGB-Farbton (rot, gelb, blau). Das Ergebnis ist die Mischung aus diesen drei "Grundfarben". Jede Ziffer kann die Werte 0 bis 9 und A bis F annehmen, wobei letztere die Werte 10 bis 15 repräsentieren. FF entspricht dem Wert 255. Die erste Ziffer muss für die Dezimalzahl mit 16 multipliziert werden. Dadurch ergibt sich 15*16 und für das hintere Zeichen 15. Ein dunkles Blau für ein Tabellenzelle wird mit `<tr bgcolor="#000088">` angegeben.

# 2. Einführung in PHP

## 2.1 Vorbemerkung und grundlegende Befehle

PHP ist leicht zu erlernen, da es keine zu großen Anforderungen an die Syntax stellt. Die meisten Fehler werden auftauchen, wenn Sie vergessen, dass jeder Befehl mit einem Semikolon abzuschließen ist. Der sog. Parser – der Teil von PHP, der das Programm für die Bildschirmdarstellung aufbereitet – meldet dann einen Fehler und das Programm wird nicht ausgeführt. Sie werden wahrscheinlich am Anfang viel Zeit benötigen, um Tippfehler auszubessern.

PHP ist C-ähnlich. Es gibt zwei wichtige Programmiersprachfamilien, die in den vielen imperativen Programmiersprachen vorherrschend sind. PASCAL- und C-ähnliche Sprachen.

Beispiel:

in Pascal-ähnlichen Sprachen:

```
procedure hallo;
begin
  Wert1:='Hallo';
end;
```

in c-ähnlichen Sprachen

```
function hallo()
{
  Wert1="Hallo";
}
```

Mit PHP werden keine selbstständig lauffähigen Programme erstellt, sondern PHP-Skripte werden mit einem Editor erstellt und abgespeichert. Immer, wenn die Datei aufgerufen wird sorgt ein sog. Parser für die Übersetzung und Ausführung des Programms.

PHP-Dateien laufen nur in Webbrowsern, es ist also eine Websprache.

HTML haben wir schon kennen gelernt. PHP und HTML arbeiten zusammen, d.h. in einem PHP-Skript kommen sowohl HTML-Zeilen als auch PHP-Code vor.

Anders als z.B. Javascript ist PHP eine sog. serverseitige Programmiersprache. Es läuft zusammen mit einem Webserver auf einem Computer. Der Webserver gibt i.d.R. ein eindeutiges Verzeichnis frei, auf das jedermann zugreifen kann.

Das bedeutet aber auch, dass jedes Skript, das wir schreiben nur in diesem Verzeichnis läuft. In unserem Beispiel handelt es sich um das Verzeichnis: c:\xampp\htdocs\Shop.

Wir werden zunächst v.a. nur den einfachen Befehl zum Ausgeben von Text benötigen. Dafür wird der echo-Befehl verwendet. Alle Befehle in PHP enden mit einem Semikolon. Nach dem Wort „echo“ wird der auszugebende Text in Anführungszeichen angegeben.

Beispiele:

a) `echo "Hallo Welt";`

b) `echo "Dies ist ein Text";`

c) `echo "Hallo", " wie", " geht es dir?";`

Wir schreiben unsere Befehle in ein HTML-Dokument und geben diesem Dokument die Endung ".php". Damit der Rechner weiß, welche Anweisungen HTML und welche PHP sind müssen alle PHP-Anweisungen innerhalb sog. PHP-Tags stehen.

PHP-Tags können überall im Dokument geöffnet und geschlossen werden. Sie haben folgendes Aussehen:

Öffnendes PHP-Tag: `<?php`

Schließendes PHP-Tag: `?>`

Beispiel:

```
<html>
<head></head>
<body>
  <p>
    <h1>Meine Ueberschrift</h1>
    <?php echo "Hallo Welt"; ?>
  </p>

  <p>
    <h1>Meine Ueberschrift</h1>
    <?php
      echo "Hallo Welt";
      echo "</p>";
    ?>
</body>
</html>
```

Beide Absätze in diesem Beispiel haben dasselbe Ergebnis bzw. Aussehen. Im ersten Absatz ist der schwarze Text als PHP-Anweisung implementiert, im zweiten auch das Absatz-Ende-Tag `</p>`. HTML und PHP lassen sich also leicht verschränken.

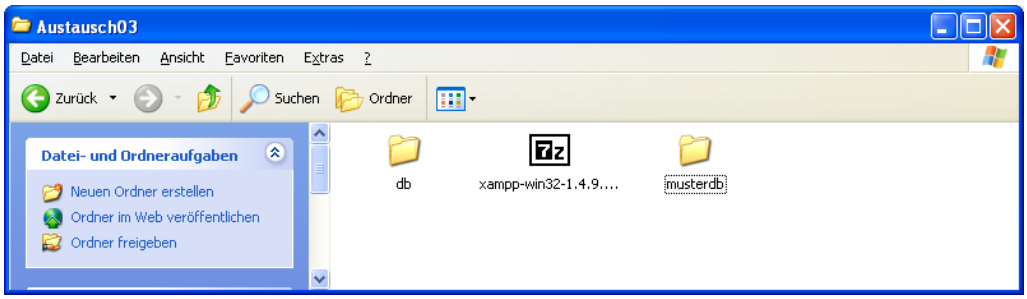
## 2.2 Installation der Software

PHP-Skripte können nur verwendet werden, wenn auf dem Computer, auf dem sich das PHP-Dokument befindet ein Webserver und der PHP-Parser gestartet ist. Die PHP-Skripte können nur in Webbrowsern ausgeführt werden. Bevor der Webserver die Datei an den Browser sendet, übersetzt der Parster die PHP-Befehle.

Die Übersetzung gibt er an den Browser weiter, der die Seite aufgerufen hat.

Der Anwender, der ein PHP-Skript aufruft, erkennt nicht, ob es sich um ein reines HTML-Dokument handelt oder ob in dem Dokument PHP-Befehle ausgeführt werden.

Im Unterricht verwendet man am besten das Programmpaket XAMPP. Die Installationsdatei kann kostenfrei unter http://apachefriends.org heruntergeladen werden.



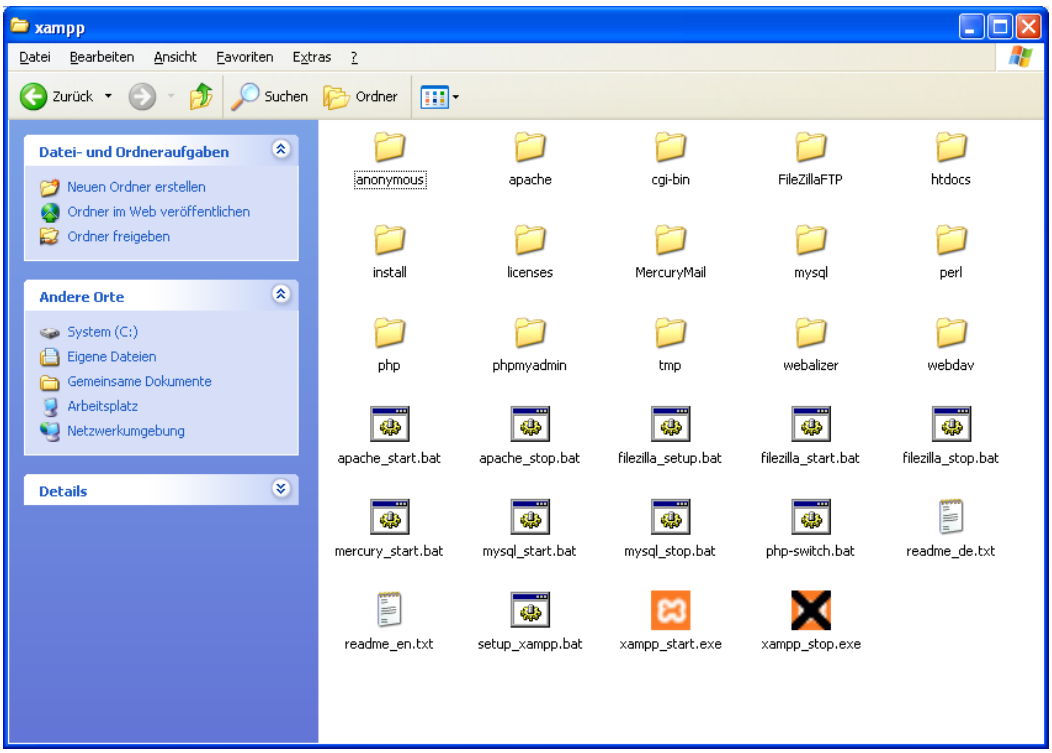

Starten Sie die Datei **xampp-win32-1.6.8...** durch Doppelklick.

Das Programm wird dann fragen, wohin Sie installieren wollen, geben Sie z.B. die Festplatte **C:** an, indem Sie

eintragen und auf **Extract** drücken.


Das „Extracting“ dauert etwa 1 Minute. Danach befindet sich auf Ihrer Festplatte C: der Ordner c:\xampp mit zahlreichen Unterverzeichnissen.

Öffnen Sie dieses Unterverzeichnis, z.B. mit dem Dateimanager.




Führen Sie einmalig die Datei **setup_xampp.bat** durch Doppelklick aus. Nach wenigen Momenten ist das System vollständig eingerichtet.

Starten Sie xampp durch Drücken der Datei **xampp_start.exe** und beenden Sie es stets mit der Datei **xampp_stop.exe**. Beenden Sie das Programm bitte nicht durch Drücken des „Kreuzes“ am rechten oberen Fensterrand.

Während Sie mit XAMPP arbeiten, darf das Programmfenster nicht geschlossen werden, Sie können es aber minimieren.

Die selbst geschriebenen Skripte müssen in das Verzeichnis c:\xampp\htdocs\ abgelegt werden. Damit man sie leicht findet am besten in einem eigenen Ordner, z.B. **c:\xampp\htdocs\Shop**

Es muss xampp gestartet sein, damit PHP laufen kann. Eine Datei, z.B. "start.php", die Sie im Verzeichnis **c:\xampp\htdocs\Shop** abgespeichert haben, starten Sie im Browser, indem Sie **127.0.0.1/Shop** in die Adresszeile eingeben.

## 2.3 Variablen und Felder (Arrays)

Programmiersprachen kommen ohne Variablen nicht aus. Variablen sind "Platzhalter" für Werte.

Man stelle sich eine Schachtel vor, etwa eine Zigarrenschachtel. Diese habe außen auf dem Deckel einen eindeutigen Namen stehen. In der Zigarrenschachtel liegt ein Zettel, auf dem ebenfalls ein Wert steht. Genau so funktioniert eine Variable. Sie hat einen eindeutigen Namen (Bezeichner) und einen Wert.

In vielen Programmiersprachen muss man eine Variable deklarieren, d.h. bevor wir mit ihr arbeiten können müssen wir festlegen, von welchem Typ (Zahl oder Wort) die Variable ist. Nicht so in PHP.

Man erkennt Variablen in PHP an dem **"$"**-Zeichen.

Ohne Deklaration kann man sofort eine Variable verwenden, indem man ihr mit dem **"="**-Zeichen einen Wert zuweist.

Beispiele:

```
$MeinName="Friedrich";
$MeinAlter=38;
$Irgendeinname="Müller";
```

Merke:
Zahlenwerte einer Variablen werden ohne Anführungs- und Schlusszeichen zugewiesen, alle Worte (alphanumerischen Zeichenketten) stehen in Anführungs- und Schlusszeichen.

Der Bezeichner der Variable kann beliebig gewählt werden. Um nicht allzu viel schreiben zu müssen verwendet man oft nur kurze Worte, z.B. $i, $b, $B. Beachten Sie, dass Groß- und Kleinschreibung unterschieden wird.

In obigen Beispielen ist als `$MeinName` der Bezeichner und "Friedrich" der Wert der Variablen.

Im Programmverlauf kann der Wert einer Variablen verändert werden, indem ihr einfach ein neuer Wert zugewiesen wird, auch hier wieder mit dem "="-Zeichen. Der Zettel in der Schachtel wird also ersetzt durch einen neuen Zettel mit einem neuen Wert.

Beispiele:

```
$Name="Meier";
$Name="Huber";
```

Der Wert der Variablen Huber ist also "Huber". Die erste Zuweisung geht durch die zweite verloren.

Welchen Wert hat die Variable $i am Ende des Programms?

```
$i=10;
$i=15;
$i=$i+10;
$i=$i*10;
```

`$i++;` ist gleichbedeutend mit der Anweisung `$i=$i+1;`

An diesem Beispiel sieht man sehr schön, dass das **"="-Zeichen** kein **"ist gleich"-Zeichen** ist, sondern eine **Wertzuweisung** darstellt. Man sollte besser sagen, `$i` "erhält den neuen Wert". Der Rechner arbeitet dabei immer von rechts nach links, d.h. er kennt

den alten Wert von `$i` noch, bevor er ihn neu vergibt. So ergibt `$i=$i+10;` keine Fehlermeldung. Der alte Wert von `$i` von 15 wird um 10 erhöht und dann erst der Variablen `$i` neu zugeordnet.

Wir haben also am Ende des Programms eine Variable `$i` mit dem Wert von 250. In unserer bildlichen Vorstellung wurde am Anfang eine Schachtel mit `$i` bezeichnet und der Zettel darin hatte den Wert 10, dann den Wert 15, dann 25 und zuletzt 250.

Variablen lassen sich verwenden, um den Inhalt auszugeben oder mit ihnen zu rechnen. Sie stellen Platzhalter dar.

Beispiel:

```
$Name="Hans Mueller";
$Klasse="6c";
$Note1=1; $Note2=4; $Note3=1;
$Durchschnittsnote=($Note1+$Note2+$Note3)/3;
echo "Hallo $Name, du besuchst die Klasse $Klasse. ";
echo "Die Durchschnittsnote in den Kernfächern betraegt $Durchschnittsnote";
```

## Konkatenation

Darunter versteht man das Zusammensetzen von Zeichenketten.

Beispiel:

```
$Vorname="Hans";
$Nachname="Müller";
```

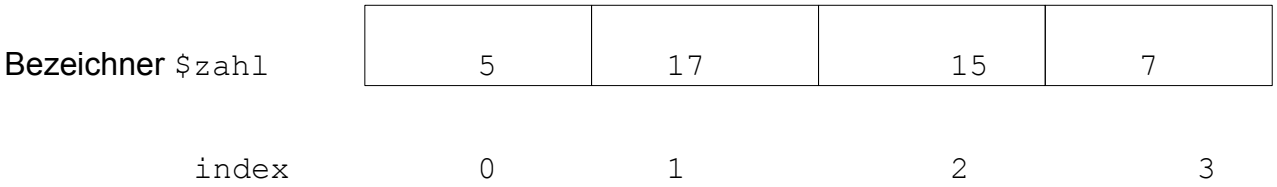
Beide Variablen sollen zu eine Variablen $Name verbunden werden. Dabei benötigen wir noch ein Leerzeichen zwischen den beiden Bestandteilen:

```
$Name=$Vorname." ".$Nachname;
```

In vielen Programmiersprachen verwendet man ein "+"-Zeichen, in PHP den "." als Konkatenations-Operator.

## Felder (Arrays)

Anstelle von vier Variablen `$Zahl1=5; $Zahl2=17; $Zahl3=15; $Zahl4=7;` kann es besser sein, die vier Werte in ein sog. Feld (Array) zu schreiben.




Das Array wird genauso einfach erstellt wie eine Variable. Als Unterscheidungsmerkmal wird in eckigen Klammern der index angegeben.

```
$Zahl[0]=5;
$Zahl[1]=17;
$Zahl[2]=15;
$Zahl[3]=7;
```

Der Vorteil von Feldern ist die Tatsache, dass sich Felder in Schleifen ansprechen lassen! (Schleifen werden im nächsten Kapitel 2.4 erläutert.)

Beispiel:

```
for ($i=0; $i<5; $i++){
  echo "Zahl:", $i+1, " ist", $Zahl[$i];
}
```

## Die Count-Funktion

Programmiersprachen zeichnen sich dadurch aus, dass mehr oder weniger viele Funktionen existieren, die der Programmierer verwenden kann. In PHP existiert eine Funktion mit der man die Länge eines Arrays ermitteln kann, die Funktion **count()**.

```
echo count($zahl);  //Ergebnis: 4
```

Damit lässt sich das Beispiel von oben so umgestalten, dass immer das gesamte Feld ausgegeben wird!

```
<?php
  $zahl[0]=7;
  $zahl[1]=5;
  $zahl[2]=3;
  $zahl[3]=17;

  for ($i=0;$i<count($zahl);$i++)
  {
    echo $zahl[$i],", ";
  }

  // Ergebnis: 7, 5, 3, 17
?>
```

## 2.4 Wiederholungen (for-Schleife mit fester Anzahl, while-Schleife als bedingte Wiederholung)

Programmiersprachen kommen ohne Schleifen nicht aus. Schleifen sind Wiederholungs-Anweisungen. Ein Text wird z. B. 100 mal geschrieben.

In Robot Karol haben die Schülerinnen und Schüler in der Jahrgangsstufe 7 Wiederholungen bereits kennengelernt.

Beispiel:

```
wiederhole 10 mal
   schritt
*wiederhole
```

In PHP sieht das Konstrukt anders aus und kann auch mehr, es verwendet eine Variable, die einen Start- und einen Endwert hat und bei jeder Wiederholung ihren Wert verändert.

Als Variable kann ein beliebiger Name herangezogen werden, z.B. `$i`. Wir geben der Variable beispielsweise den Startwert 0, also

```
$i=0;
```

Wir definieren, wie oft die Wiederholung ausgeführt werden soll. Dazu bestimmen wir zuerst, wie sich der Wert von `$i` bei jeder Wiederholung verändern soll. I.d.R. wird er sich um 1 erhöhen.

```
$i=$i+1;
```

Damit haben wir eine Zählvariable, die sich immer um 1 erhöht.

Zum Schluss müssen wir noch festlegen, wie oft die Wiederholung erfolgen soll, z. B. 10 mal. Da wir bei 0 beginnen werden wir die Wiederholung solange durchführen, solange `$i` noch kleiner als 10 ist, also

```
$i<10;
```

Alle drei Werte, "Startwert", "Abbruchbedingung" und "Veränderung des Startwertes" werden nun benötigt, um eine Wiederholung zu implementieren:

```
for ($i=0; $i<10; $i=$i+1) echo "Hallo";
```

Soll mehr als ein Befehl wiederholt werden, so sind die Befehle in geschweifte Klammern zu setzen.

```
for ($i=0; $i<10; $i=$i+1){
  echo "Hallo";
  echo "<br>";
}
```

### Beispiele:

Wie oft werden die Anweisungen aufgeführt?

```
for ($i=5; $i<10; $i=$i+1) echo "Hallo";
```

Start mit 5, dann Wiederholumg mit 6, 7, 8, 9 also 5 mal

```
for ($i=5; $i<10; $i=$i+2) echo "Hallo";
```

Start mit 5, dann 7, dann 9 also 3 mal

```
for ($i=0; $i<100; $i=$i+10) echo "Hallo";
```

Start mit 0, dann 10, 20, 30, 40, 50, 60, 70, 80, 90 also 10 mal

Die Zählvariable, in unseren Beispielen `$i` kann in den Befehl mit integriert werden wie jede andere Variable auch:

```
for ($i=0; $i<10; $i=$i+1) echo "Dies ist die $i. Zahl<br>";
```

Das Ergebnis sieht dann so aus:

```
Dies ist die 0. Zahl
Dies ist die 1. Zahl
Dies ist die 2. Zahl
Dies ist die 3. Zahl
Dies ist die 4. Zahl
Dies ist die 5. Zahl
Dies ist die 6. Zahl
Dies ist die 7. Zahl
Dies ist die 8. Zahl
Dies ist die 9. Zahl
```

Möchte man, dass die Ausgabe mit 1 beginnt so hat man zwei Möglichkeiten, entweder man lässt die For-Schleife mit 1 beginnen und bis `$i<=10` zählen oder man verändert die echo-Anweisung:

Alternative 1:

```
for ($i=1; $i<=10; $i=$i+1) echo "Dies ist die $i. Zahl<br>";
```

Alternative 2 (zur Berechnung mit Konkatenation):

```
for ($i=0; $i<10; $i=$i+1) echo "Dies ist die".($i+1)." Zahl<br>";
```

## Bedingte Wiederholung (while-Schleife)

Manchmal weiß man beim Erstellen eines Programms nicht, wie oft eine Anweisung wiederholt werden soll.

Beispiel:

```
$SatzNr = 1;
  while (odbc_fetch_row($Ergebnis, $SatzNr)) {
    $Spalte1[] = odbc_result($Ergebnis, 1);
    $Spalte2[] = odbc_result($Ergebnis, 2);
    $Spalte3[] = odbc_result($Ergebnis, 3);
    $Spalte4[] = odbc_result($Ergebnis, 4);
    $SatzNr++;
  }
```

In der Variable $Ergebnis ist das Ergebnis einer `SELECT`-Abfrage mit vier Spalten an eine Datenbank gespeichert. Das Ergebnis der Abfrage soll nun in vier Arrays zwischengespeichert werden. Die Anzahl der Zeilen ist aber von der aktuelle Datenbank abhängig. Die Bedingung `odbc_fetch_row` ist solange wahr, bis das Ende der Abfrage erreicht ist.

Auf die Anbindung einer Datenbank mit PHP wird in Kapitel 2.8 eingegangen.

# 2.5 Funktionen (Methoden)

*Das Kapitel 2.5 ist für den Online-Shop nicht zwingend erforderlich. Da Methoden aber zu den grundlegenden Bestandteilen von Programmiersprachen gehören, soll es in diesem Tutorial nicht fehlen, v.a. weil Schülerinnen und Schüler mit Programmiererfahrung diese selbstverständlich verwenden werden.*

Aus Gründen

1. der Übersichtlichkeit
2. der Wiederverwendbarkeit von Code
3. der geringeren Schreibarbeit
4. der geringeren Fehleranfälligkeit

von Programmen teilt man ein Programm in verschiedene Anweisungsblöcke auf. In den Programmiersprachen werden verschiedene Wörter für diese Anweisungsblöcke verwendet, die wir der Einfachheit halber alle synonym verwenden wollen: Prozeduren, Funktionen, Anweisungen, Methoden.

Die Schüler haben in der Jahrgangsstufe 7 in der Software "Robot Karol" Anweisungen bereits kennengelernt:

Beispiel:

```
anweisung umdrehen
   linksdrehen
   linksdrehen
*anweisung
```

Der Name der Anweisung, hier "umdrehen" kann beliebig gewählt werden. Ist eine Anweisung definiert, kann im Hauptprogramm die Funktion einfach durch Verwendung des Namens aufgerufen werden.

In PHP ändert sich an der Definition von Funktionen nichts Grundlegendes. Anstelle von Anweisung verwendet man das Schlüsselwort `function`. Der Name ist wiederum beliebig, jedoch schließt der Name immer mit runden Klammern ab: `()`. Anstelle eines Ende-Zeichens werden die Befehle einfach in geschweifte Klammern gesetzt.

Beispiel:

```
function Einladung(){
   echo "Hallo Anton<br>";
   echo "Hiermit lade ich dich ein am 10.10.2004<br>";
   echo "zu meiner Geburtstagsfeier.<br>";
   echo "Viele liebe Gruesse<br>";
   echo "Dein Hans Dampf";
}
```

Unser PHP-Code wird meist zusammen mit HTML-Code gemischt Die Funktionen definieren wir am Besten am Ende, also nach dem `</HTML>`-Tag. Beachten Sie, dass auch alle Funktionen zu PHP gehören und daher innerhalb von PHP-Tags stehen müssen.

Der Aufruf der Funktionen erfolgt aber z. B. innerhalb des `<BODY>`-Tags. Auch dabei handelt es sich um PHP-Befehle, also benötigen wir auch hier die PHP-Tags.

Beispiel:

Der graue Text ist HTML, der orange sind die PHP-Tags, das blaue der Aufruf der Methode und der Name der Funktion:

```
<html><head><title>Meine Erste Funktion</title></head>
```

```
<body>
<p>Hier steht ganz normaler HTML-Text<br>
In der naechsten Zeile beginnt der Funktionsaufruf<br>
<?php Einladung(); ?>
Durch das Ende des PHP-Tags ist dies wieder normaler HTML-Code.
</body>
</html>


<?php
function Einladung(){
   echo "Hallo Anton<br>";
   echo "Hiermit lade ich dich ein am 10.10.2004<br>";
   echo "zu meiner Geburtstagsfeier.<br>";
   echo "Viele liebe Gruesse<br>";
   echo "Dein Hans Dampf";
}
?>
```

Die Funktion `Einladung()` ist nicht besonders spannend. Sinnvoll lassen sich Funktionen oft anwenden, wenn Variablen existieren. Schauen wir uns auch hierzu ein Beispiel an, das sich an obigem orientiert. Zu Beginn werden in einem PHP-Tag zwei Variablen festgelegt, `$Name` und `$Datum`. Die Funktion `Einladung()` verändern wir auch in der Art, dass sie nur mit zwei Variablen aufgerufen werden kann. Die verlangten Variablen (sog. Parameter) stehen in den runden Klammern. In der Funktion dürfen die Variablen andere Namen haben. Sie bekommen beim Aufruf im Hauptprogramm auf jeden Fall die richtigen Werte übergeben, man spricht von Parameter-Übergabe.

```
<?php
$Name="Anton"; $Datum="10.10.2004";
?>

<html><head><title>Meine Erste Funktion</title></head>
<body>
<p>Hier steht ganz normaler HTML-Text<br>
In der naechsten Zeile beginnt der Funktionsaufruf<br>
<?php Einladung($Name, $Datum); ?>
Durch das Ende des PHP-Tags ist dies wieder normaler HTML-Code.
</body>
</html>

<?php
function Einladung($Var1, $Var2){
   echo "Hallo $Var1<br>";
   echo "Hiermit lade ich dich ein am $Var2<br>";
   echo "zu meiner Geburtstagsfeier.<br>";
   echo "Viele liebe Gruesse<br>";
   echo "Dein Hans Dampf";
}
?>
```

Hinweis:
Es ist nicht möglich, eine Funktion in der oben beschriebenen Art ohne Variablenübergabe auszuführen. Variablen, die im Hauptprogramm deklariert wurden, sind in der Funktion nämlich unbekannt.

Funktionen können auch einen Rückgabewert haben, also ein Ergebnis. Gekennzeichnet werden diese Ergebnisse durch das Schlüsselwort `return`. Der Vorteil einer solchen Implementierung liegt u.a. daran, dass die Funktion selbst wie eine Variable verwendet werden kann.

## Beispiel:

```
$Ergebnis=BerechneFlaeche(10,20);
echo $Ergebnis;

echo BerechneFlaeche(10,20);

fucntion BerechneFlaeche($Laenge, $Breite){
 return $Laenge * $Breite;
}
```

## 2.6 Bedingte Anweisungen

Bedingungen sind Ausdrücken, die entweder `wahr` oder `falsch` sind. Mit einer bedingten Anweisung (if-Anweisung) kann die Ausführung einer Anweisung von der Bedingung abhängig gemacht werden.

Beispiel:

```
$Geschlecht="m"; $Name=Meier;

if ($Geschlecht=="m")
{
  echo "Sehr geehrter Herr ", $Name;
} else
{
  echo "Sehr geehrte Frau, $Name;
}
```

Die Bedingung nach dem Schlüsselwort `if` steht in Klammern. Der nachfolgende Befehl bzw. der nachfolgende Anweisungsblock wird nur ausgewertet, wenn wenn die Bedingung `wahr` ist. Die Angabe des `else`-Teils ist nicht erforderlich.

Der Ausdruck kann mit

| | |
|---|---|
| < | kleiner als |
| > | größer als |
| <= | kleiner oder gleich |
| >= | größer oder gleich |
| == | ist gleich |
| <> | ist ungleich |

erstellt werden.

Mehrere Bedingungen können mit und-Operatoren (`&&`) bzw. oder-Operatoren (`||`) verknüpft werden.

## 2.7 Formular-Daten verwalten

Betrachten wir das folgende PHP-Programm "start.php", in dem ein Formular verwendet wird.

```
<html><head></head>
<body>
  <form action="start.php" method="GET">
    <input type="text" name="wert1"><br>
    <input type="text" name="wert2"><br>
    <input type="text" name="wert3"><br>
    <input tpye="submit">
  </form>
</body>
</html>
```

Das Formular besteht aus drei Text-Eingabefeldern und einem Button. Wird der Button gedrückt, dann wird das Ziel "antwort.php" aufgerufen und dieser Seite werden die drei eingetragenen Werte als wert1, wert2 und wert3 mit übertragen.

Die Seite "antwort.php" soll so aussehen:

```
<html><head></head>
<body>
  echo $wert1."/".$wert2."/".$wert3." ergibt in der Summe: ".($wert1+$wert2+$wert3);
</body>
</html>
```

Im obigen Beispiel werden die Variablen `$wert1`, `$wert2` und `$wert3` verwendet. Diese Variablen gibt es aber noch nicht. Das Formular verfügt nur über die HTML-Werte: `wert1`, `wert2` und `wert3` in der Adresszeile. Wir benötigen also eine Verbindung zwischen HTML-Werten und PHP-Variablen. Wir benötigen drei PHP-Variablen, die den Wert der HTML-Werte annehmen. Dann kann die Funktion mit diesen PHP-Variablen aufgerufen werden:

PHP stellt eine Funktion zur Verfügung, wie Variablen den Wert eines HTML-Formulars annehmen können.

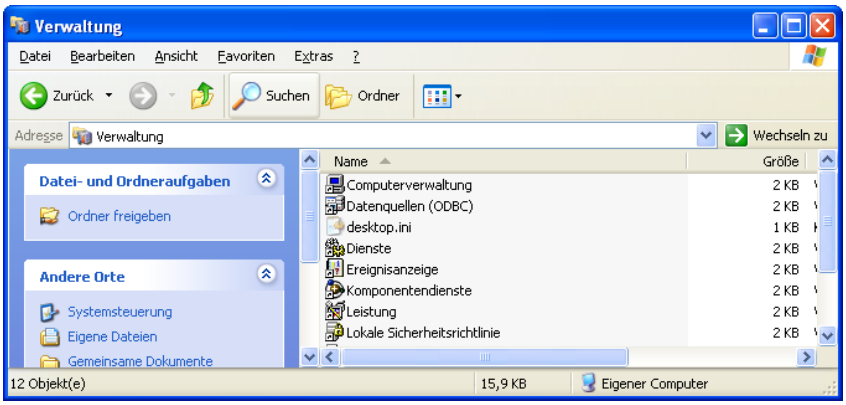
```
<?php
  $wert1=$_GET["wert1"];
  $wert2=$_GET["wert2"];
  $wwert3=$_GET["wert3"];
?>
```

Es werden also wie in PHP üblich neue Variablen deklariert, diese erhalten den Wert, den das Formular mit der Methode GET erhalten hat. Diesen PHP-Quellcode tragen wir ganz oben - noch vor dem HTML-Teil - in das Programm "start.php" ein. Dann funktioniert das Programm und berechnet die Summe der in das Formular eingegebenen Werte.

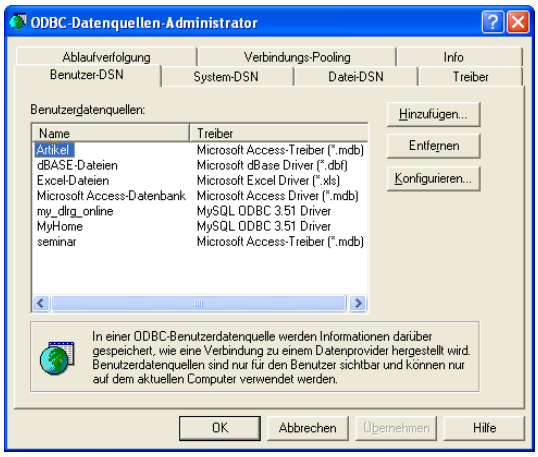
Hinweis:
Eine Seite kann sich auch selbst aufrufen. Im Form-Tag tragen wir dann unter action denselben Namen ein, die die Seite hat, also z. B. "start.php". Wenn wir die Seite abschicken, öffnet sie sich ein zweites Mal. Allerdings erkennen wir in der Adresszeile des Browsers, dass die Werte `wert1`, `wert2` und `wert3` mit übergeben werden.

## 2.8 ODBC-Anbindung einer Datenbank

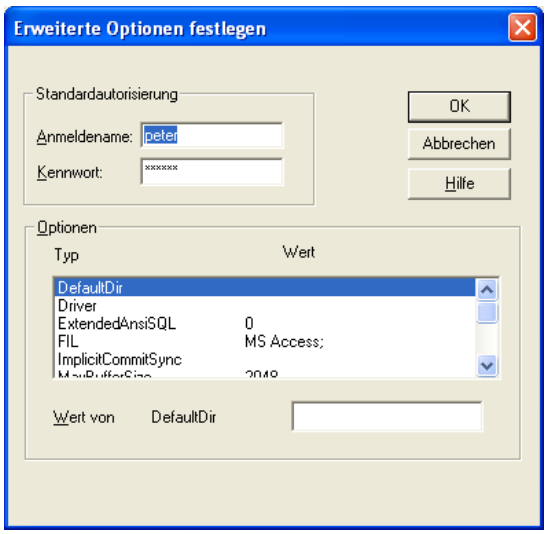
*Open Database Connectivity* ist ein Standard, um von Programmiersprachen auf eine Datenbank zugreifen zu können, ohne dass das DBMS gestartet sein muss. Für die gängigen Datenbanken gibt es ODBC-Treiber, die im Betriebssystem installiert werden. Unter Windows befinden sich die ODBC-Verbindungen im Ordner Verwaltung innerhalb der Systemsteuerung.




Für eine beliebige Datenbank, z. B. eine MS Access Datenbank, kann hier eine ODBC-Schnittstelle eingerichtet werden. Dazu wählt man den Eintrag `Hinzufügen` aus, wählt den Datenbank-Treiber aus und gibt unter `Konfigurieren|Datenbank auswählen` den Pfad zur Datenbank an und unter `Erweitert` noch einen Login-Namen und ein Passwort für den Zugriff auf die Datenbank.







## 2.9 SQL-Befehle mit PHP ausführen

Wenn Sie MS Access auf ihrem Computer installiert haben, dann sollte der entsprechende ODBC-Treiber bereits vorhanden sein. Für andere Datenbanken lassen sich die Treiber von den Download-Seiten der Hersteller herunterladen und auf ihrem Betriebssystem installieren. Welche Datenbank Sie dann verwenden ist unabhängig von der Programmierung des E-Shop-Programms.

Der Zugriff auf die Datenbank erfolgt in PHP in wenigen Schritten. Zunächst müssen sie nur den Namen der ODBC-Schnittstelle, den Login-Namen und das Passwort kennen. Bauen Sie mit `odbc_connect(...)` ein Verbindung zur Datenbank auf und starten Sie einen SQL-Befehl mit dem Befehl `odbc_exec(...)`. Das Ergebnis der Abfrage wird in einer Variable gespeichert.

Beispiel:

```
$ODBC_Name = "Artikel";
$Benutzer = "peter";
$Passwort = "geheim";
$SQLString = "SELECT ArtikelNr, Artikelname, Preis, Lieferzeit FROM Artikel;"
$Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
$Ergebnis = odbc_exec($Quelle, $SQLString);
```

Mithilfe der Bedingten Wiederholung (While-Schleife) kann das Ergebnis der Abfrage zeilenweise ausgelesen werden. Die Anweisung `odbc_result($Ergebnis, 1)` holt den Inhalt der ersten Spalte der aktuell ausgewählten Zeile aus der Ergebnistabelle. Bei jeder Wiederholung wird eine neue Zeile der Ergebnistabelle ausgelesen. Dazu muss innerhalb der Schleife der Wert der Variable `$SatzNr` um eins erhöht werden. In diesem Fall werden vier Attribute jeweils in ein dazu gehörendes Array kopiert.

```
$SatzNr = 1;
while (odbc_fetch_row($Ergebnis, $SatzNr)) {
  $Spalte1[] = odbc_result($Ergebnis, 1);
  $Spalte2[] = odbc_result($Ergebnis, 2);
  $Spalte3[] = odbc_result($Ergebnis, 3);
  $Spalte4[] = odbc_result($Ergebnis, 4);
  $SatzNr++;
}
```

SQL-Befehle zum Verändern von Datensätzen (UPDATE), zum Löschen von Datensätzen (DELETE) und zum Einfügen von Datensätzen (INSERT) liefern einen Wahrheitswert zurück. Eine Auswertung erfolgt in unserem Web-Shop aus Gründen der Übersichtlichkeit des Programmcodes allerdings nicht.

## 2.10 Session-Verwaltung mit PHP

Ein grundlegendes Problem von HTML war lange Zeit die Zustandslosigkeit von HTML. In HTML ist es serverseitig nicht möglich zu erkennen, ob ein Benutzer schon einmal auf einer Seite war. HTML kennt keine Variablen. So ist es schwierig zu entscheiden, welcher Nutzer gerade auf eine HTML-Seite zugreift.

Oftmals wurde und wird das Problem mit sogenannten Cookies gelöst, Informationen die auf der Client-Seite abgelegt werden. Seit der Version 4.0 besitzt PHP eine Session-Verwaltung, mit der das Problem leichter gelöst werden kann.

Mithilfe der Anweisung `session_start()` erzeugt der Server eine eindeutige ID bzw. übernimmt die für diese Seite. Sie ist so lange, dass sie nicht erraten werden kann. Jede Seite des E-Shops beginnt mit dem Befehl `session_start()`. Bei jedem Zugriff auf die Seite wird diese ID an den Server gesendet und so kann das Programm entscheiden, welcher Nutzer gerade auf die Seite zugreift. Es lassen sich zudem beliebig viele Variablen als Session-Variablen speichern. Im E-Shop wird z. B. die Kaufsumme der bestellten Artikel in dieser Variable gespeichert und kann dann an beliebiger Stelle im Text ausgegeben werden.

Immer wenn der Browser geschlossen wird ist die Session-ID gelöscht. Nach Abschluss eines Kaufvorgangs kann die Session-ID aber auch mithilfe des Befehls `session_regenerate_id();` neu erzeugt werden. Den aktuellen Wert der Session-ID kann mit dem Befehl `session_id();` ermittelt werden. So lassen sich die aktuellen Kundendaten mit den Bestelldaten in der Datenbank aufnehmen und zuordnen.

# 3. Der Online-Shop

## 3.1 Die Anzeige der Artikelgruppen – die Datei "start.php"

Die Datei `start.php` wird – wie de Name sagt, zum Beginn des Online-Shops aufgerufen. Später kann die Datei auch in index.php umbenannt werden, dann wird das Dokument automatisch gestartet, sobald der die Ordneradresse aufgerufen wird. Es werden alle Artikelgruppen angezeigt und in Klammern die Anzahl der Artikel in jeder Gruppe.

Zu Beginn des Dokumentes wird die Session-Verwaltung gestartet.

```
<?php
  session_start();
?>
```

Normaler Kopf einer HTML-Datei.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<title>Der P-Seminar-Shop</title>
</head><body>
```

Eine SQL-Abfrage aller Artikelgruppen aus der ODBC-Datenbank.

```
<?
   $ODBC_Name = "Artikel";
   $Benutzer = "peter";
   $Passwort = "geheim";
   $SQLString = "SELECT Artikelgruppe, count(Artikelname)FROM Artikel GROUP BY
      → Artikelgruppe ORDER BY Artikelgruppe";
   $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
   $Ergebnis = odbc_exec($Quelle, $SQLString);
```

Das Ergebnis in zwei Arrays `$Wert[]` und `$Zahl[]` kopieren.

```
   $SatzNr = 1;
   while (odbc_fetch_row($Ergebnis, $SatzNr)) {
      $Wert[] = odbc_result($Ergebnis, 1);
      $Zahl[] = odbc_result($Ergebnis, 2);
      $SatzNr++;
   }
```

Die Anzahl der Gruppen nochmals in einer Variable speichern.

```
    $Anzahl = count($Wert);
?>
```

Jetzt wird eine Tabelle aufgebaut, in der mittleren Spalte werden die Artikelgruppen angezeigt.

```
<table border="0" width="100%">
  <tr>
    <td width="100%" bgcolor="#000088" colspan="4">
      <h2 align="center"><font color="#FFFFFF"> Der P-Seminar-Shop </font></h2>
    </td>
  </tr>

  <?php
  for ($n = 0; $n < $Anzahl; $n = $n + 1 )
  {
    ?>
    <tr>
      <td width="33%" bgcolor="#FFFFFF"> </td>
      <td width="33%" bgcolor="#FFFFFF" align="center">
```

Der Link erhält eine Variable Gruppe, dessen Wert die anzuzeigende Artikelgruppe

bestimmt. Klickt der Anwender auf den Link wird die Datei „Artikelliste.php“ aufgerufen.

```
         <a href="Artikelliste.php?Gruppe=<? echo $Wert[$n] ?>">
            <?php
              if ($Wert[$n])
              {
                 echo $Wert[$n], " (", $Zahl[$n], ")";
              }
            ?>
         </a>
      </td>

      <td width="33%" bgcolor="#FFFFFF"> </td>
    </tr>
    <?
}
?>

<tr bgcolor="#000088">
    <td width="33%" height="10"> </td>
    <td width="33%"> </td>
    <td width="33%"> </td>
</tr>
</table>

</body>
</html>
```

## 3.2 Die Anzeige einer Liste aller Artikel einer Artikelgruppe – die Datei „Artikelliste.php“

Mit `session_start();` wird die Session-Verwaltung weitergeführt. Die Variable `$Gruppe` erhält den Wert aus der Adresszeile.

```
<?php
  session_start();
  $Gruppe=$_GET['Gruppe'];
?>

<html><head>
<title>Der Shop - Artikelliste</title>
</head><body bgcolor="#FFFFFF">

<?php
```

Wird die Seite aufgerufen, ohne dass eine Gruppe angegeben wurde, so wird automatisch die Gruppe „Sonstiges“ ausgewählt. Die Funktion `!isset($Gruppe)` prüft, ob es eine Variable `$Gruppe` gibt.

```
   if (!isset($Gruppe)) {
     $Gruppe = "Sonstiges";
   }
```

Die Datenbank-Abfrage, die alle Artikel einer Gruppe auswählt.

```
   $ODBC_Name = "Artikel";
   $Benutzer = "peter";
   $Passwort = "geheim";
   $SQLString = "SELECT ArtikelNr, Artikelname, Preis, Lieferzeit FROM Artikel WHERE
      → Artikelgruppe ='$Gruppe' ORDER BY ArtikelNr";
   $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
   $Ergebnis = odbc_exec($Quelle, $SQLString);
```

Das Ergebnis der Abfrage wird in vier Arrays kopiert, für jede Spalte der Ergebnis-Tabelle ein Array.

```
   $SatzNr = 1;
   while (odbc_fetch_row($Ergebnis, $SatzNr)) {
      $Spalte1[] = odbc_result($Ergebnis, 1);
      $Spalte2[] = odbc_result($Ergebnis, 2);
      $Spalte3[] = odbc_result($Ergebnis, 3);
      $Spalte4[] = odbc_result($Ergebnis, 4);
      $SatzNr++;
   }
?>
```

In der folgenden Tabelle werden die Artikel aufgelistet. Zuerst werden in der ersten Zeile die Überschriften erstellt.

```
<table border="0" width="100%">
  <tr>
    <td width="100%" bgcolor="#000088" colspan="5">
      <h2 align="center"><font color="#FFFFFF">
         <?php echo $Gruppe; ?>
         </font>
      </h2>
    </td>
  </tr>
  <tr bgcolor="#000088">
    <td width="10%"><font color="#FFFFFF">
        ArtikelNr</font></td>
```

```
    <td width="35%"><font color="#FFFFFF">
        Artikelbezeichnung</font></td>
    <td width="15%" align="center">
        <font color="#FFFFFF">
        Preis (Euro)</font></td>
    <td width="20%" align="center">
        <font color="#FFFFFF">
        Lieferzeit</font></td>
    <td width="20%" align="center">
        <font color="#FFFFFF">
        Details</font></td>
    <td width="20%"></td>
  </tr>
```

Die folgende Wiederholung listet alle Artikel auf. Die Daten stammen aus den vier erstellen Arrays. Mit `count($Spalte1)` wird die Anzahl der Datensätze erfragt. Die Farbe wechselt in jeder Zeile. Dazu wird die laufende Nummer mit dem Modulo-Operator geprüft. Der Modulo-Operator gibt den Rest einer Ganzzahlendivision an, hier wird durch 2 geteilt, das bedeutet das Ergebnis ist 1 für eine ungerade Zahl und 0 für eine gerade Zahl.

```
<?
   for ($n = 0; $n <= count($Spalte1)-1; $n++ ) {
       if (($n % 2) == 1) {
          $Farbe = "#888888";
       }
       else {
          $Farbe = "#FFFFFF";
       }
?>

  <tr bgcolor="<?php echo $Farbe ?>">
    <td width="10%" align="center">
        <?php echo $Spalte1[$n] ?>
    </td>
    <td width="35%">
        <?php echo $Spalte2[$n] ?>
    </td>
    <td width="15%" align="right">
        <?php echo number_format($Spalte3[$n], 2, ",", ".") ?>
    </td>
    <td width="20%">
        <?php echo $Spalte4[$n] ?>
    </td>
    <td width="20%" align="center">
```

Der Link auf die Seite „Artikel.php“ führt zur detaillierten Anzeige des ausgewählten Artikels.

```
       <a href="Artikel.php?ArtikelNr=<?php
                echo trim(strval($Spalte1[$n])),
               "&Gruppe=", $Gruppe ?>">
          Details...
       </a>
    </td>
  </tr>
<?  } ?>
```

Zum Schluss noch eine abschließende leere Zeile und ein paar Links.

```
  <tr bgcolor="#000088">
    <td width="10%" height="10"> </td>
    <td width="35%"> </td>
    <td width="15%"> </td>
    <td width="20%"> </td>
    <td width="20%"> </td>
  </tr>
</table>

<table border="0" width="100%">
```

```
  <tr>
    <td width="20%">
       <a href="start.php">
          Artikelgruppen
       </a>
    </td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%">
    <?php for ($n = 1; $n <= $Seiten; $n++) { ?>
       <a href="artikelliste.php?Gruppe=<?php
         echo $Gruppe ?>">
         [ <?php echo $n ?> ]
       </a>
    <?php } ?>
    </td>
    <td width="20%">
    </td>
  </tr>
</table>

</body>
</html>
```

## 3.3 Die Warenpräsentation und die Bestellung – die Datei "artikel.php"

Diese Seite bringt eine detaillierte Anzeige des ausgewählten Artikels. Auch kann der Anwender hier die Anzahl der zu bestellenden Artikel eingeben und abschicken.

Es beginnt wieder mit der Session-Verwaltung.

```
<?php
  session_start();
  $session_id=session_id();
```

In der Varialbe `$ArtikelNr` ist die ID des Aritikels gespeichert, die angezeigt werden soll. Mithilfe eines einfachen SQL-Befehls können alle Informationen aus der Datenbank gelesen werden.

```
  $ArtikelNr=$_GET['ArtikelNr'];
?>

<html><head>
<title>Der P-Seminar-Shop - Artikelanzeige</title>
</head><body bgcolor="#FFFFFF">

<?php
  // Daten der ODBC-Datenbank auf dem Windows-Rechner
  $ODBC_Name = "Artikel";
  $Benutzer = "peter";
  $Passwort = "geheim";
  // Abfrage des Artikels mit der Richtigen Artikel-Nummer
  $SQLString = "SELECT ArtikelNr, Artikelname, Beschreibung,Preis, Artikelgruppe, Bild
       → FROM Artikel WHERE ArtikelNr = $ArtikelNr";
  $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
```

Das Ergebnis der Abfrage wird in der Variable `$Ergebnis` gespeichert.

```
  $Ergebnis = odbc_exec($Quelle, $SQLString);
```

Die Ergebnistabelle hat nur eine Zeile, die Spalten werden in Variablen zwischengespeichert.

```
  odbc_fetch_row($Ergebnis,1);
  // Es werden nun nacheinander alle Spalten angezeigt
  $ArtikelNr = odbc_result($Ergebnis, 1);
  $Artikelname = odbc_result($Ergebnis, 2);
  $Beschreibung = odbc_result($Ergebnis, 3);
  $Preis = odbc_result($Ergebnis, 4);
  $Artikelgruppe = odbc_result($Ergebnis, 5);
  $Bild = odbc_result($Ergebnis, 6);
?>

<table border="0" width="100%">
  <tr>
    <td width="100%" bgcolor="#000088">
      <h2 align="center"><font color="#FFFFFF"><?php echo $Artikelname ?> </font></h2>
    </td>
  </tr>
</table>
```

Die Artikel können mit einem Bild versehen werden. In der Datenbank ist nur der Bildname gespeichert. Für den Fall, dass es kein Bild für einen Artikel gibt, soll ein Standardbild mit dem Namen `KeinBild.jpg` verwendet werden.

```
<table border="0" align="center" width="80%">
  <tr>
    <td align="center" width="29%" rowspan="5">
```

```
      <?php if (!file_exists($Bild)) {$Bild = "KeinBild.jpg"; }
      ?>
      <img src="<?php echo $Bild ?>" width="200" height="150" alt="kein Bild">
    </td>
    <td width="71%"> <?php echo $Artikelname ?> </td>
  </tr>
  <tr>
    <td width="71%">
```

In PHP kann eine Zahl formatiert ausgegeben werden mit der Anzahl der Nachkommastellen und beliebigen Komma- und Tausenderzeichen.

```
      Preis: <?php echo number_format($Preis, 2, ",", ".") ?> Euro
    </td>
  </tr>
  <tr>
    <td width="71%"> <?php echo $Beschreibung ?> </td>
  </tr>
  <tr>
    <td width="71%"> <?php echo $Artikelgruppe ?> </td>
  </tr>
  <tr>
    <td width="71%">
          <form action="warenkorb.php" method="get">
          <input name="Zeit" type="hidden"
                 value="<?php echo time(); ?>">
```

In diesen versteckten Eingabefeldern werden die Artikeldaten hinterlegt.

```
          <input name="ArtikelNr" type="hidden"
                 value="<?php
                 echo $ArtikelNr; ?> ">
          <input name="Artikelname" type="hidden"
                 value="<?php
                 echo trim($Artikelname); ?> ">
          <input name="Preis" type="hidden"
                 value="<?php
                 echo $Preis; ?> ">
          <input name="Gruppe" type="hidden"
                 value="<?php echo $Artikelgruppe ?> ">
          Menge: <input name="Menge" type="text">
          <input type="submit" value="In den Warenkorb damit">
        </form>
    </td>
  </tr>
  <tr>
    <td width="100%" colspan="2"></td>
  </tr>
</table>

<table border="0" width="100%">
  <tr bgcolor="#000088">
    <td width="10%" height="10"></td>
    <td width="35%"></td>
    <td width="15%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>

  <tr>
    <td width="20%">
       <a href="start.php">
          Artikelgruppen
       </a>
    </td>
    <td width="20%">
       <a href="artikelliste.php?Gruppe=<?php echo $Gruppe ?> ">
          Zurueck zur Artikelauswahl
       </a>
    </td>
    <td width="20%"></td>
    <td width="20%"></td>
```

```
    <td width="20%"></td>
  </tr>
</table>

</body>
</html>
```

## 3.4 Die Anzeige des Warenkorbs – die Datei "warenkob.php"

Diese Datei ist wohl die schwierigste im ganzen Projekt. Es sollen alle bestellten Artikel mit ihren Preisen angezeigt werden. Diese Seite wird angezeigt, wenn ein Artikel bestellt wurde, daher ist es Aufgabe dieser Seite, den zuletzt bestellten Artikel in die Datenbank aufzunehmen. Außerdem soll es möglich sein, dass der Anwender einzelne Artikel auch wieder löschen kann.

Alle Artikeldaten der Bestellung werden in Variblen übernommen. Bei der ArtikelNr wird vorsorglich die Funktion `trim($ArtieklNr)` aufgerufen, die vorhandene Leerzeichen am Ende löscht. Falls eine Variable `loeschen` gesendet wurde, so wird auch diese übernommen. Das ist dann der Fall, wenn der Anwender im Warenkorb auf den entsprechenden Link klickt.

```
<?php
  session_start();
  $session_id=session_id();
  $Zeit=$_GET['Zeit'];
  $ArtikelNr=$_GET['ArtikelNr'];
  $Artikelname=$_GET['Artikelname'];
  $Preis=$_GET['Preis'];
  $Gruppe=$_GET['Gruppe'];
  $Menge=$_GET['Menge'];
  $ArtikelNr=trim($ArtikelNr);
  $loeschen=$_GET['loeschen'];
```

Wenn der Wert der Variable $loeschen 1 ist, dann wird mithilfe des entsprechenden SQL-Befehls der bestellte Artikel in der Datenbank gelöscht.

```
  if ($loeschen==1) {
    $ODBC_Name = "Artikel";
    $Benutzer = "peter";
    $Passwort = "geheim";
    $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
    $SQLString = "DELETE From Bestellung WHERE session_id='$session_id' AND
      →ArtikelNr='$ArtikelNr'";
    $Ergebnis = odbc_exec($Quelle, $SQLString);
  }
```

Sonst wird der zuletzt ausgewählte Artikel mit seiner Anzahl in der Datenbank gespeichert. Die Session-ID gilt als Identifikationsmerkmal des Users.

```
  else {
    $ODBC_Name = "Artikel";
    $Benutzer = "peter";
    $Passwort = "geheim";
    $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
```

Durch Betätigen des Vor- und Zurück-Buttons des Browsers kann nicht ausgeschlossen werden, dass eine Bestellung mehrmals hintereinander ausgeführt wird. Aus diesem Grund erfolgt zuerst eine Sicherheitsabfrage, die prüft, ob der Artikel mit der aktuellen Artikel-Nummer schon von diesem Anwender bestellt wurde. In diesem Fall wird der alte Eintrag mithilfe des SQL-Befehls gelöscht.

```
    $SQLString = "SELECT ArtikelNr FROM Bestellung WHERE session_id='$session_id' AND
      →ArtikelNr='$ArtikelNr'";
    $Ergebnis = odbc_exec($Quelle, $SQLString);

    $SatzNr=1;
    while (odbc_fetch_row($Ergebnis, $SatzNr)) $SatzNr++;
```

Wenn die Variable `$SatzNr` noch den Wert 1 hat, dann gibt es den Artikel noch nicht. Die Bestellung kann gespeichert werden.

```
    if ($SatzNr==1)
```

```
    $SQLString = "INSERT into Bestellung (session_id, ArtikelNr, Artikelname, Preis,
      →Gruppe, Menge, Zeit, Bestaetigt) VALUES ('$session_id', '$ArtikelNr',
      →'$Artikelname', '$Preis','$Gruppe', '$Menge', '$Zeit', 'nein');";
```

Sonst existiert schon eine Bestellung dieses Artikels – der Eintrag wird mithilfe des folgenden SQL-Befehls ersetzt.

```
    else $SQLString="Update Bestellung SET session_id='$session_id',
      →Artikelname='$Artikelname', Preis='$Preis', Gruppe='$Gruppe', Menge='$Menge',
      →Zeit='$Zeit', Bestaetigt='nein' WHERE ArtikelNr='$ArtikelNr';";

    $Ergebnis = odbc_exec($Quelle, $SQLString);
  }
?>
```

Jetzt kann mit der Anzeige der in der Datenbank schon vorhandenen Artikel, die bestellt wurden, begonnen werden.

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
<title>P-Seminar-Shop - Warenkorb</title>
</head>
<body>

  <table border="0" width="100%">
    <tr>
      <td width="100%" bgcolor="#000088" colspan="7">
        <h2 align="center"><font color="#FFFFFF">
          P-Semianr-Shop Warenkorb </font></h2>
      </td>
    </tr>

    <tr bgcolor="#000088">
      <td width="10%"></td>
      <td width="10%"><font color="#FFFFFF">
          ArtikelNr</font></td>
      <td width="25%"><font color="#FFFFFF">
          Bezeichnung</font></td>
      <td width="10%" align="center"><font color="#FFFFFF">
          Einzelpreis in EUR</font></td>
      <td width="10%" align="center"><font color="#FFFFFF">
          Menge</font></td>
      <td width="15%" align="center"><font color="#FFFFFF">
          Gesamtpreis (Euro)</font></td>
      <td width="10%"></td>
    </tr>

    <?php
```

Abfrage aller Artikel im Warenkorb.

```
    $SQLString = "SELECT ArtikelNr, Artikelname, Preis, Menge FROM Bestellung WHERE
      →((session_id='$session_id') AND (Bestaetigt='nein'));";
    $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
    $Ergebnis = odbc_exec($Quelle, $SQLString);
```

Kopieren aller Einträge der Ergebnistabelle in vier Arrays.

```
    $SatzNr = 1;
    while (odbc_fetch_row($Ergebnis, $SatzNr)) {
      $Spalte1[] = odbc_result($Ergebnis, 1);
      $Spalte2[] = odbc_result($Ergebnis, 2);
      $Spalte3[] = odbc_result($Ergebnis, 3);
      $Spalte4[] = odbc_result($Ergebnis, 4);
      $SatzNr++;
    }

    $Summe = 0;
    $Zahl = count($Spalte1);
```

Anzeige aller betellten Artikel.

```
     for ($nr = 0; $nr < $Zahl; $nr++) {
       $Summe = $Summe + ($Spalte3[$nr] * $Spalte4[$nr]);
  ?>

  <tr>
    <td width="10%"></td>
    <td width="10%"> <?php echo $Spalte1[$nr] ?> </td>
    <td width="25%"> <?php echo $Spalte2[$nr] ?> </td>
    <td width="10%" align="center"> <?php echo number_format($Spalte3[$nr],2,",",".") ?>
</td>
    <td width="10%" align="center"> <?php echo $Spalte4[$nr] ?></td>
    <td width="15%" align="right">
      <? echo number_format($Spalte3[$nr] * $Spalte4[$nr],2,",",".") ?>
    </td>
```

In dieser Zelle wird ein Link auf die Eigene Seite `warenkorb.php` gesetzt. Hier erfolgt die Angabe des Variablenwertes `loeschen=1` mit der dazugehörenden `ArtikelNr`.

```
    <td width="10%" align="center">
       <a href="warenkorb.php?loeschen=1&ArtikelNr=<?php echo $Spalte1[$nr] ?
      → >">l&ouml;schen</a>
    </td>
  </tr>
```

Die Summe, die oben berechnet wird kann als Session-Variable gespeichert werden. Damit steht sie später auch noch zur Verfügung. Ansonsten folgt, wir sind ja noch in der Wiederholungsschleife die Ausgabe der Artikeldaten, die sich im Warenkorb befinden.

```
<?php } $_SESSION['Summe']=$Summe; ?>

  <tr bgcolor="#FFFFFF">
    <td width="10%" height="10"></td>
    <td width="10%"></td>
    <td width="25%"></td>
    <td width="10%"></td>
    <td width="10%">Rechnungssumme (inkl. MwSt) = </td>
    <td width="15%" align="right">
    <b> <?php echo number_format($Summe, 2, ",", ".") ?> <b> </td>
    <td width="10%"></td>
  </tr>
</table>


<table border="0" width="100%">
  <tr bgcolor="#000088">
    <td width="10%" height="10"></td>
    <td width="35%"></td>
    <td width="15%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%">
       <a href="start.php">
          Artikelgruppen
       </a>
    </td>
    <td width="20%">
       <a href="artikelliste.php?Gruppe=<? echo $Gruppe ?>">
          Zurueck zur Artikelauswahl
       </a>
    </td>
    <td width="20%">
    <?php if ($Zahl > 0) { ?>
       <a href="kundendaten.php?Summe=<? echo $Summe ?>">
          Zur Kasse
       </a>
    <?php } ?>
    </td>
```

```
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>
</table>

</body>
</html>
```

## 3.5 Die Eingabe der Kundendaten – die Datei "kundendaten.php"

In der vorletzten Datei muss der Anwender noch seine Daten angeben. Dazu wird hier ein einfaches Formular verwendet. Diese Datei ist sehr einfach und verweist dann im Attribut auf die letzte Datei `danke.php`.

```
<?php
  session_start();
?>

<html><head><title>Der Shop Kundendaten</title>
</head><body>
<table border="0" width="100%">
  <tr>
    <td width="100%" bgcolor="#000088" colspan="4">
      <h2 align="center"><font color="#FFFFFF">
          Kundendaten
      </font></h2>
    </td>
  </tr>
</table>

<form action="Danke.php">
<table border="0" width="100%">
  <tr>
    <td width="20%"></td>
    <td width="13%">Name:</td>
    <td width="27%">
        <input name="Name" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">Vorname:</td>
    <td width="27%">
       <input name="Vorname" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">Strasse:</td>
    <td width="27%">
       <input name="Strasse" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">PLZ:</td>
    <td width="27%">
       <input name="Plz" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">Ort:</td>
    <td width="27%">
       <input name="Ort" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">Telefon:</td>
```

```
    <td width="27%">
       <input name="Telefon" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%">E-Mail:</td>
    <td width="27%">
       <input name="EMail" type="text" size="35">
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="20%"></td>
    <td width="13%"></td>
    <td width="27%">
       <input type="hidden"
              name="Summe"
              value="<?php echo $Summe ?>">
       <input type="submit"
              value="Bestellung abschicken" >
       <input type="reset"
              value=" Loeschen " >
    </td>
    <td width="20%"></td>
  </tr>
</table>
</form>

<table border="0" width="100%">
  <tr bgcolor="#000088">
    <td width="20%"> </td>
    <td width="20%"> </td>
    <td width="20%"> </td>
    <td width="20%"> </td>
    <td width="20%"> </td>
  </tr>
</table>

<table border="0" width="100%">
  <tr>
    <td width="20%">
       <a href="JavaScript:history.back()">
          Zurück
       </a>
    </td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>
</table>

</body>
</html>
```

## 3.6 Der Abschluss der Bestellung – die Datei "danke.php"

In dieser Datei werden die Daten, die von der Datei kundendaten.php gesendet werden noch in der Datenbank gespeichert. Das Attribut `session_id` bildet den Fremdschlüssel der Tabelle `Kunde` zur Tabelle `Bestellung`.

Zu Beginn werden die übersandten Daten in Variablen übernommen und dann in der Datenbank gespeichert.

```
<?php
  session_start();
  $session_id=session_id();
  $Name=$_GET['Name'];
  $Vorname=$_GET['Vorname'];
  $Strasse=$_GET['Strasse'];
  $Plz=$_GET['Plz'];
  $Ort=$_GET['Ort'];
  $Telefon=$_GET['Telefon'];
  $EMail=$_GET['EMail'];
  $Summe=$_SESSION['Summe'];

  // hier wird die Bestellung in der Datenbank gespeichert
  // Daten der ODBC-Datenbank auf dem Windows-Rechner
  $ODBC_Name = "Artikel";
  $Benutzer = "peter";
  $Passwort = "geheim";
  $SQLString = "INSERT into Kunde (Name, Vorname, Strasse, Plz, Ort, Telefon, EMail,
       → Summe, session_id) VALUES ('$Name', '$Vorname', '$Strasse', '$Plz','$Ort',
       → '$Telefon', '$EMail', '$Summe','$session_id');";
  $Quelle = odbc_connect($ODBC_Name, $Benutzer, $Passwort);
  $Ergebnis = odbc_exec($Quelle, $SQLString);
  $SQLString = "UPDATE Bestellung SET Bestaetigt='Bestaetigt' WHERE
       → session_id='$session_id';";
  $Ergebnis = odbc_exec($Quelle, $SQLString);
  session_regenerate_id();
?>

<html><head><title>Der P-Seminar-Shop - Dank</title>
</head><body>
```

Zum Abschluss erfolgt noch die Anzeige, dass die Bestellung erfolgreich angekommen ist. Dazu wird neben den Adressdaten auch nochmals die Summe der Bestellung angezeigt.

```
<table border="0" width="100%">
  <tr>
    <td width="100%" bgcolor="#000088" colspan="5">
      <h2 align="center"><font color="#FFFFFF">
          Der P-Seminar Shop sagt Dankeschön!
      </font></h2>
    </td>
  </tr>
</table>

<table border="0" width="100%">
  <tr>
    <td width="30%"></td>
    <td width="50%">
       Sehr geehrte(r) Herr/Frau, <?php echo $Name,"," ?>
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="30%"></td>
    <td width="50%">
       Wir bedanken uns für Ihre Bestellung.
       Die Waren gehen an folgende Adresse:
    </td>
    <td width="20%"></td>
```

```
  </tr>
  <tr>
    <td width="30%"></td>
    <td width="50%">
       <b>
          <?php echo "Herr/Frau ", $Vorname, " ", $Name; ?> <br>
          <?php echo $Strasse; ?> <br>
          <?php echo $Plz, " ", $Ort;  ?>
       </b>
    </td>
    <td width="20%"></td>
  </tr>
  <tr>
    <td width="30%"></td>
    <td width="50%">
       Die Zahlung des Rechnungsbetrags von
       <?php echo number_format($Summe, 2, ",", ".") ?>
       warten wir ab. Nach Zahlungseingang werden die Artikel versendet.
    </td>
    <td width="20%"></td>
  </tr>
</table>


<table border="0" width="100%">
  <tr bgcolor="#000088">
    <td width="20%" height="10"></td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>
</table>

<table border="0" width="100%">
  <tr>
    <td width="20%">
       <a href="start.php">Zur Startseite</a></td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
    <td width="20%"></td>
  </tr>
</table>


</body>
</html>
```