13 Unterprogramme erstellen

»Non prendere il lavoro come un nemico, e non farne nemmeno l'unica ragione della tua vita. – Betrachte die Arbeit nicht als Feind und mache sie auch nicht zum einzigen Grund deines Lebens.« Nehmen Sie sich das zu Herzen und erleichtern Sie sich die Arbeit mit VBA, indem Sie Ihre Programme in kleine logische Einheiten unterteilen! Solche Einheiten werden als Unterprogramme bezeichnet.

Wie die Arbeit mit Unterprogrammen funktioniert, möchte ich Ihnen in diesem Kapitel zeigen. Unterprogramme sind vor allem dann sehr nützlich, wenn es Anweisungen in einem Programm gibt, die mehrmals aufgerufen werden, oder wenn Sie dieselben Anweisungen in verschiedenen Prozeduren benötigen. Dann kann man diese Anweisungen in ein Unterprogramm »auslagern«. Ein solches Unterprogramm kann dann entweder mehrfach in einer Prozedur aufgerufen werden oder im zweiten Fall können verschiedene Prozeduren dasselbe Unterprogramm aufrufen.

Grundsätzlich unterscheidet man zwei Arten von Unterprogrammen: Function- und Sub-Prozeduren. Function-Prozeduren (oder kurz Funktionen genannt) werden meist für Berechnungen eingesetzt und haben einen Ergebniswert. Sub-Prozeduren führen häufig eine Reihe von Anweisungen aus, ohne dabei einen Wert zu berechnen und zurückzugeben.

Globale Function- und Sub-Prozeduren

Benötigen Sie die gleichen Function- und Sub-Prozeduren in verschiedenen Formularen, so erfassen Sie die Unterprogramme in einem eigenen Modul und machen sie so allgemein zugänglich. Legen Sie dazu mit EINFÜGEN ➧ MODUL ein entsprechendes Modul an.

13.1 Function-Prozeduren

Eine Function-Prozedur besteht aus einer Reihe von Anweisungen, die zwischen einer `Function`- und einer `End Function`-Anweisung stehen. Sie rufen eine solche Function-Anweisung aus einer Prozedur heraus auf, indem Sie den Namen der Function-Prozedur angeben und gegebenenfalls zusätzlich – in Klammern – einige Parameter, die die Function-Prozedur steuern.

13.1.1 Sie kennen bereits Function-Prozeduren!

Im Prinzip haben Sie Function-Prozeduren bereits kennen gelernt, sie wurden von mir bisher immer kurz als Funktionen bezeichnet. So kennen Sie bereits die Funktion `Now` (die Systemdatum und -zeit Ihres PCs zurückgibt) oder die Funktion `Format` (die eine Zahl nach bestimmten Angaben formatiert als Zeichenkette zurückgibt).

Die Funktion `Now` wird ohne Parameter verwendet, mit der Zeile

```
datZeit = Now
```

wird die Funktion `Now` aufgerufen, die Systemdatum und -zeit bestimmt und diese an die Variable `datZeit` übergibt. Wie die Funktion `Now` im Einzelnen aussieht, wissen wir nicht. Da sie in VBA integriert ist, haben wir darauf auch keinen Einfluss.

Die Funktion `Format` verwendet im Gegensatz zur `Now`-Funktion zwei Argumente. Mit der Zeile

```
strDM = Format(txtEuro * dblKurs, "#,##0.00")
```

wird die Funktion `Format` aufgerufen und es werden ihr beim Aufruf gleichzeitig zwei Argumente mit übergeben: Zum einen ein Wert (hier in Form einer Multiplikation), der formatiert werden soll, zum anderen die Formatierungsvorschrift. Die Funktion gibt dann die berechnete und formatierte Zeichenkette an die Variable `strDM` zurück.

13.1.2 So sieht eine Function-Prozedur aus

Ganz allgemein können Sie eine Function-Prozedur durch die folgenden Zeilen beschreiben:

```
[Public | Private] Function Name [(Argumentenliste)] [As Typ]
    [Anweisungen]
    [Name = Ausdruck]
    [Exit Function]
    [Anweisungen]
    [Name = Ausdruck]
End Function
```

Werden mehrere Argumente in der Argumentenliste genannt, werden sie durch Kommata voneinander getrennt. Zudem sollten Sie jedem Argument der Argumentenliste einen Datentyp zuweisen, sonst werden die Argumente als Datentyp `Variant` übergeben.

Beispielsweise kann die folgende erste Zeile einer Function-Prozedur für ein zu übergebendes Argument so:

```
Private Function MwSt(dblBetrag As Double) As String
```

oder für zwei Argumente so:

```
Private Function MwSt(dblBetrag As Double, _
        intSatz As Integer) As String
```

aussehen. Neben den Argumenten, die an die Funktion übergeben werden, wird auch der Rückgabewert der Funktion deklariert. In obigem Beispiel wird der Wert für `MwSt` als `String` deklariert.

Funktion zum Errechnen der Mehrwertsteuer

Stellen Sie sich vor, Sie arbeiten in einer Prozedur mit Nettobeträgen und möchten dazu in einer Funktion die Mehrwertsteuer berechnen. Sie erstellen dazu eine Function-Prozedur, der zwei Argumente übergeben werden, nämlich der Nettobetrag, für den die Mehrwertsteuer errechnet werden soll, und der Mehrwertsteuersatz, der 0, 7 oder 16 Prozent betragen kann.

```
Private Function MwSt(dblBetrag As Double, _
                      intSatz As Integer) As String
    MwSt = Format(dblBetrag * intSatz / 100, "#,##0.00")
End Function
```

13.1.3 So rufen Sie eine Function-Prozedur auf

Soll die Funktion `MwSt` in einer Prozedur aufgerufen werden, so könnte der Aufruf beispielsweise

```
strSteuer = MwSt(2345, 16)
```

lauten oder

```
strSteuer = MwSt(dblNetto, intSteuerschlüssel)
```

oder auch

```
MsgBox "Die Mehrwertsteuer beträgt " & _
        MwSt(dblNetto, intSteuersatz) & " DM."
```

13.1.4 Beispiel: Eigene Funktion definieren

Sie haben die Möglichkeit, beliebige eigene Funktionen zu definieren. Für die Casa Maria habe ich eine Funktion ausgedacht, die angibt, wieviel ein Appartement pro Quadratmeter pro Person pro Tag kostet. Dazu teile ich den Preis pro Tag durch die Quadratmeter und durch die maximale Anzahl der Personen, wobei ich die Personenzahl um 0,5 erhöhe, wenn ein Kleinkind zusätzlich erlaubt ist. Diese Spezialfunktion für das Formular *frmAppartements* sieht dann so aus:

```
Option Compare Database
Option Explicit

Private Function AppKennzahl(dblPreis As Double, _
                 dblFläche As Double, dblPers As Double, _
                 fKind As Boolean) As Double

If fKind = True Then
    dblPers = dblPers + 0.5
End If

AppKennzahl = Format(dblPreis / (dblFläche * dblPers),"0.00")

End Function
```

Aufgerufen wird die Funktion über das Klicken auf die Schaltfläche CMDAPPKENNZAHL, die ich neu zum Formular *frmAppartements* hinzugefügt habe:

```
Private Sub cmdAppKennzahl_Click()
Dim dblDEM As Double, dblEUR As Double

dblDEM = AppKennzahl(Me.PreisDEM, Me.Größe, _
        Me.MaximaleBelegung, Me.KleinkindZusätzlichErlaubt)
dblEUR = AppKennzahl(Me.PreisEUR, Me.Größe, _
        Me.MaximaleBelegung, Me.KleinkindZusätzlichErlaubt)

MsgBox dblDEM & " DM/Tag/Pers/qm   oder  " & _
        dblEUR & " `/Tag/Pers/qm "

End Sub
```

Mit `Me` Felder des aktuellen Formulars ansprechen

Mit `Me` können Sie schnell und bequem die Felder des aktuellen Formulars ansprechen. Wenn Sie den Punkt nach `Me` eingegeben haben, bekommen Sie automatisch eine Liste mit Auswahlmöglichkeiten angezeigt.

Damit das Ganze in der beschriebenen Form funktioniert, habe ich für das Formular *frmAppartements* als neue Datensatzherkunft die Abfrage *qryAppartementsMitPreis* festgelegt. Darin habe ich aus der Tabelle tblPreisgruppen das Feld PREISPROTAG als `PreisDEM` eingebunden und das Feld `PreisEUR` berechnet. Dabei habe ich auf den schon aus Kapitel 3 bekannten Ausdruck `Runden(PreisProTag/1,95583,0)` zurückgegriffen. Beide Felder habe ich als Zusatzinformation in das Formular *frmAppartements* mit eingebunden.

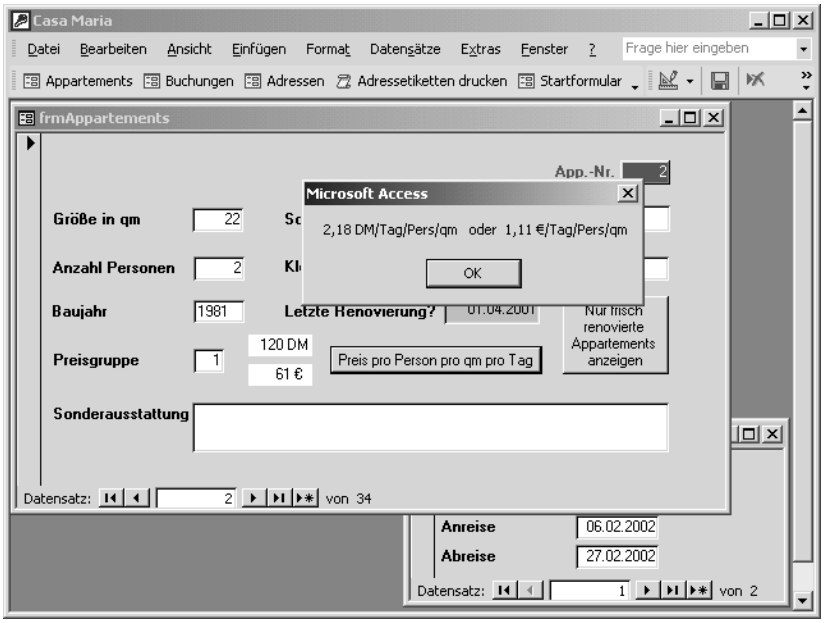

Bild 13.1:
Die selbst definierte Function-Prozedur wurde aufgerufen

13.2 Sub-Prozeduren

Der Hauptunterschied von Sub-Prozeduren zu Function-Prozeduren besteht darin, dass eine Sub-Prozedur keinen Wert zurückgibt. Ansonsten wird sie ebenso wie eine Funktion über ihren Namen aufgerufen und Sie können einer Sub-Prozedur beim Aufruf ebenso Argumente übergeben.

Allgemein lässt sich eine Sub-Prozedur durch

```
[Private | Public] Sub Name [(Argumentenliste)]
    [Anweisungen]
    [Exit Sub]
    [Anweisungen]
End Sub
```

beschreiben. Sollen mehrere Argumente übergeben werden, werden sie durch Komma voneinander getrennt. Geben Sie – wie in Funktionen auch – für die einzelnen Argumente deren Datentyp an, sonst werden sie als Argumente vom Datentyp `Variant` behandelt.

Der Aufruf einer Sub-Prozedur erfolgt über ihren Namen: Sollen Argumente verwendet werden, werden sie (ohne Klammern) hinter dem Namen aufgeführt, wie

```
Prozedurname txtEingabe
```

Es gibt eine zweite Möglichkeit, eine Sub-Prozedur aufzurufen, die aber nicht so häufig verwendet wird:

```
Call Prozedurname(txtEingabe)
```

Sie verwendet dabei das Schlüsselwort `Call`. Jetzt werden die Argumente – wie in einer Funktion – mit Klammern übergeben.

Beispiel: Motto des Tages für alle

Das Motto des Tages soll in den Titelleisten aller Formulare erscheinen. Dazu wird die Variable `gstrMotto` in einem eigenen Modul als globale Variable deklariert. Erstellen Sie dazu mit EINFÜGEN • MODUL das Modul `mdlMotto`. Das Abfragen des Tagesmottos wird in diesem Modul als Sub-Prozedur angelegt.

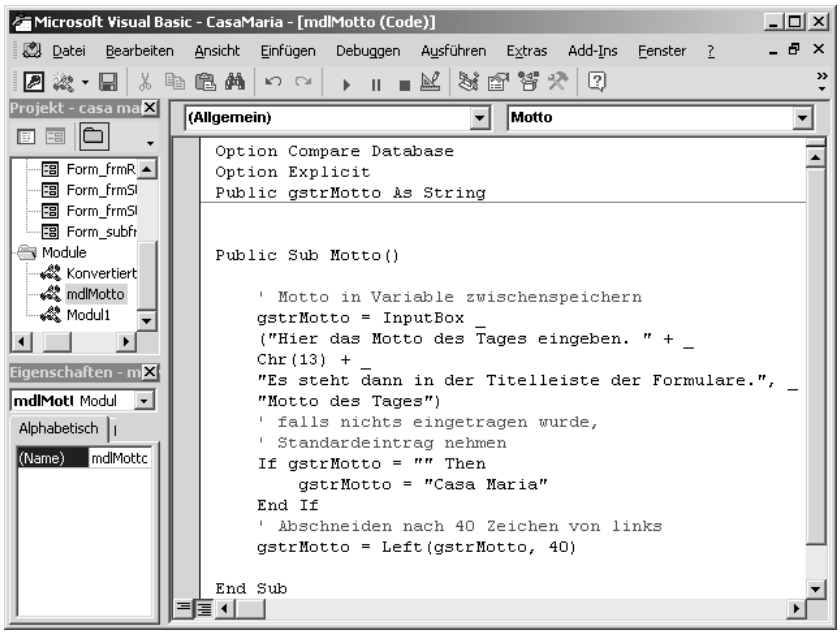

Bild 13.2: Sub-Prozedur im Modul *mdlMotto*

Mit `Chr(13)` wird im Text der InputBox ein Zeilenumbruch eingefügt. Vom eingegebenen Text werden mit `Left (gstrMotto,40)` die

ersten 40 Zeichen abgeschnitten, da für viel mehr im Startformular gar kein Platz wäre.

Die Sub-Prozedur soll jetzt nicht mehr direkt im Startformular ausgeführt werden, sondern wird von dort aus aufgerufen. Die Ereignisprozedur, die beim Öffnen des Formulars *frmStart* ausgeführt wird, ändert sich wie folgt:

```
Private Sub Form_Open(Cancel As Integer)

    Motto  ' Ruft die Sub-Prozedur auf
    ' Motto in Titelleiste des Formulars schreiben
    Me.Form.Caption = gstrMotto

End Sub
```

Mit `Me.Form.Caption = gstrMotto` wird das in der Variablen gespeicherte Motto in die Titelleiste des gerade geöffneten Formulars geschrieben.

Fügen Sie die Ereignisprozedur ohne Aufruf der Sub-Prozedur in den Code der anderen Formulare ein.

```
Private Sub Form_Open(Cancel As Integer)

    ' Motto in Titelleiste des Formulars schreiben
    Me.Form.Caption = gstrMotto

End Sub
```

Damit wird beim Öffnen des jeweiligen Formulars das Motto in die Titelleiste übernommen. Wenn Sie möchten, können Sie zusätzlich ein kleines Kürzel an den Anfang stellen, also z.B. `Me.Form.Caption = "App. "& gstrMotto` für das Formular *frmAppartements*. Damit lassen sich mehrere geöffnete Formulare in der Taskleiste besser unterscheiden.

13.3 Variablenübergabe

Zwei Verfahren der Übergabe von Argumenten an Function- und Sub-Prozeduren werden von VBA unterstützt: per Referenz und per Wert. Was verbirgt sich dahinter?

13.3.1 Übergabe per Referenz

Wenn es nicht anders angegeben ist, wird ein Argument »per Referenz« übergeben. Das folgende Beispiel zeigt die Auswirkungen einer Übergabe per Referenz. Fügen Sie die folgenden Prozeduren in ein neues Modul namens *mdlÜbergabeTest* ein.

```
Sub PerReferenz(dblBetrag As Double)
    '16% Mwst
    dblBetrag = dblBetrag * 1.16
    MsgBox "Brutto: " & dblBetrag
End Sub

Sub PerReferenzTest()
    Dim dblSumme As Double

    dblSumme = 1000
    MsgBox "Summe 1 = " & dblSumme
    PerReferenz dblSumme
    MsgBox "Summe 2 = " & dblSumme
End Sub
```

Zum Testen der Prozedur `Sub PerReferenzTest` klicken Sie irgendwo in die Prozedur und dann auf die (F5)-Taste.

Wird die Funktion `PerReferenzTest` ausgeführt, wird der Wert `1000` der Variablen `dblSumme` zugewiesen. Der Wert wird vom Programm in einer MessageBox mit dem Text `Summe 1 = 1000` eingeblendet. Im Programm wird anschließend die Prozedur `PerReferenz` mit dem Argument `dblSumme` ausgeführt. Die Prozedur rechnet das übergebene Argument zu einem Bruttowert inklusive Mehrwertsteuer um, der in einer MessageBox als `Brutto: 1160` angezeigt wird. Der letzte Befehl der Prozedur `Test` gibt die Variable `dblSumme` erneut in einer MessageBox aus. Sie zeigt den Text `Summe 2 = 1160`.

Der Aufruf der Prozedur `PerReferenz` hat also den Inhalt der Variable `dblSumme` der Prozedur `Test` verändert. Der Grund dafür ist, dass bei der standardmäßigen Übergabe per Referenz der Prozedur `PerReferenz` die Adresse im Speicher übergeben wird, an der sich die Variable `dblSumme` befindet. `PerReferenz` verwendet diese Speicheradresse, benennt diese aber `dblBetrag`. `dblSumme` und `dblBetrag` sind damit zwei verschiedene Namen für den gleichen Speicherplatz, allerdings nur während des Aufrufs von `PerReferenz`. Jede Änderung an `dblBetrag` verändert also auch `dblSumme`.

13.3.2 Übergabe per Wert

Bei der Übergabe »per Wert«, die durch das Befehlswort `ByVal` bei der Definition eines Arguments einer Prozedur definiert wird, wird nicht die Speicheradresse, sondern der Inhalt der Variable übergeben. Im nächsten Listing wurde das Beispiel von oben umgearbeitet.

```
Sub PerWert(ByVal dblBetrag As Double)
    '16% Mwst
    dblBetrag = dblBetrag * 1.16
    MsgBox "Brutto: " & dblBetrag
End Sub

Sub PerWertTestVal()
    Dim dblSumme As Double

    dblSumme = 1000
    MsgBox "Summe 1 = " & dblSumme
    PerWert dblSumme
    MsgBox "Summe 2 = " & dblSumme
End Sub
```

Durch die Übergabe per Wert wird der Inhalt der Variablen `dblSumme` in der Prozedur `PerWertTest` nicht durch den Aufruf der Prozedur `PerWert` verändert.

Übergeben Sie Ihre Variablen mit `ByVal`

Der Einsatz von `ByVal` kann Fehler vermeiden helfen, die durch unerwünschte und unbeabsichtigte Veränderungen in Prozeduren entstehen können.