Microsoft EXCEL

VBA

Visual Basic für Applikationen

FHBB, Prof. Dr. L. Lehner / I. Vilim, Muttenz, April 1998
Revidiert: August 2001

Inhaltsverzeichnis

Literatur : *Reed Jacobson* *MS-Excel 97 Visual Basic, Microsoft Press, 1997*
Patrizia Prudenzi *VBA mit Excel 97 lernen, Addison-Wesley, 1997*

Einstieg in die Makroprogrammierung

Eine der Stärken von EXCEL ist seine Makrosprache. Makros sind Programme, die dem Anwender komplizierte und zeitaufwendige Arbeiten abnehmen. Seit der ersten Excel-Version zeichnet sich Excel durch seine umfangreiche und flexible Makrosprache gegenüber anderen Tabellenkalkulationsprogrammen aus. Die damalige Makrosprache war im Vergleich zu den aktuellen Möglichkeiten ziemlich schwer zu erlernen und zu verstehen. Ab Version 5.0 benutzt Excel die Programmiersprache ***Visual Basic***. Da diese Programmiersprache um einige besondere Funktionen für Anwendungsprogramme erweitert wurde, bezeichnet man diese Sprache auch mit *Visual **B**asic für **A**pplikationen* oder kurz **VBA**. Statt dem historisch bedingten Begriff Makro verwendet man jetzt auch die Bezeichnung VBA-Programme.

Die Version **VBA**, die Bestandteil von MS-Excel ist, weist in vielen Bereichen deutliche Verbesserungen gegenüber der ersten Version auf. Die Sprache VBA steht jetzt nicht nur in Excel, sondern auch in den Programmen WinWord, Access und PowerPoint zur Verfügung. Dies schafft eine einheitliche Plattform für die Programmierung der Office-Programme.

Ein VBA-Programm, das z.B. eine allgemeine Liste von Namen sortiert, braucht nur einmal in einem der Office-Programme geschrieben zu werden und kann dann in allen anderen benutzt werden. Dagegen kann ein VBA-Programm, das in EXCEL einen Zellbereich bearbeitet, natürlich nicht ohne weiteres in WinWord fehlerfrei ablaufen.

Wenn Sie in Excel Makros schreiben wollen, müssen Sie lernen, wie Visual Basic verwendet wird. Alles, was Sie aber über Visual Basic lernen, können Sie nicht nur in Excel einsetzen, sondern auch in anderen Anwendungen, die Visual Basic unterstützen. Visual Basic erleichtert Ihnen nicht nur die Arbeit, sondern erlaubt Ihnen auch, Eigenschaften zu verwenden, auf die Sie über die Standardoberfläche sonst keinen Zugriff hätten.

Sie können Makros für folgende Aufgaben einsetzen:

- Routineaufgaben automatisieren
- Benutzerdefinierte Funktionen erstellen
- Dialogfeld-Steuerelemente in Arbeitsblättern verwenden
- Benutzerdefinierte Symbolleisten, Menüs und Befehlsschaltflächen erstellen

Ein VBA-Programm kann auf zwei Arten entstehen:

- Aufzeichnung
- Programmierung

Sie können ein Makro schreiben, indem Sie den Makro-Recorder verwenden. Der Makro-Recorder arbeitet wie ein Kassettenrecorder: Ihre Aktionen werden aufgezeichnet, um jederzeit wieder abgespielt bzw. ausgeführt werden zu können. Bei der Aufzeichnung werden die Schritte, die Sie durchführen, in die Programmiersprache Visual Basic übersetzt. Die Aufzeichnung von Programmen erfordert keinerlei Kenntnisse der Programmierung. Der Anwender muss nur wissen, wie man ein Makro startet. Dieser Weg eignet sich für einfache, immer wiederkehrende Tätigkeiten. Die Methode der Aufzeichnung ist für den Anfänger der schnellste Weg, um VBA auf die Spur zu kommen. Das erstellte Makro kann mit weiteren Befehlen der Programmiersprache ***Visual Basic*** ergänzt werden – VB enthält zahlreiche Befehle, die sich nicht aufzeichnen lassen.

Das Programm wird in einer speziellen Umgebung erstellt, die "Entwicklungsumgebung" oder auch VBE (Visual Basic Editor) genannt wird. Die direkte Programmierung erfordert die Kenntnisse der Syntax und der Sprachelemente der Sprache VBA und auch Kenntnisse über die Excel-Objekte.

Routineaufgaben automatisieren

Zuerst werden einfache Einsatzmöglichkeiten von Makros in Excel durch kleine Beispiele erläutert. Sie lernen dabei die Makroaufzeichnungsfunktion von Excel kennen. Durch Interpretation und Bearbeitung von aufgezeichneten Makros lernen Sie auch die Entwicklungsumgebung von VBA kennen.

Kopieren Sie die Datei BUDGET.XLS *in Ihr Verzeichnis.*

Die folgenden vier Beispiele zeigen die Erstellung von vier einfachen Makros zur Datei BUDGET.XLS

Empfehlung:	Beispiele zuerst ohne Makroaufzeichnung ausprobieren, dann erst aufzeichnen, sonst werden alle Fehler und ihre jeweiligen Berichtigungen mitaufgezeichnet.

1. Beispiel: ***Währungsformat zuweisen***

Erstellen des Makros

1. Starten Sie Excel, öffnen Sie **BUDGET.XLS** und unter dem Namen **NEU.XLS** speichern.
2. Markieren Sie die Zellen D7:F8 der Tabelle
3. In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Makro aufzeichnen oder* → **EXTRAS** → **MAKRO** → **AUFZEICHNEN ...**

   Die Symbolleiste Visual Basic anzeigen

   Klicken Sie mit der rechten Maustaste in eine beliebige Symbolleiste *oder* → **ANSICHT** → **SYMBOLLEISTEN**. Ein Menü, das die Namen aller Symbolleisten enthält, wird angezeigt. Wählen Sie *Visual Basic* aus.
4. Ersetzen Sie den voreingestellten Makronamen durch *FormatWährung* und OK
5. → **FORMAT** → **ZELLEN** → **ZAHLEN**. Wählen Sie Währung in der Liste Kategorie aus. Im Feld Dezimalstellen geben Sie 0 ein, dann OK.
6. In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Aufzeichnung beenden*.
7. Arbeitsmappe speichern

Beachten Sie, dass während der Makroaufzeichnung das Wort ***Aufzeich.*** in der Statuszeile angezeigt ist und die Schaltfläche **Aufzeichnung beenden** eingeblendet wird.

Ausführen des Makros

1. Wählen Sie in der Tabelle Budget97 die Zellen D9:F10 aus.
2. In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Makro ausführen*
   oder → **EXTRAS** → **MAKRO** → **MAKROS ...**
3. Makro *FormatWährung* markieren und → **AUSFÜHREN**

Dem Makro eine Tastenkombination zuweisen

Mit Hilfe einer Tastenkombination ist das Ausführen des Makros einfacher.

1. In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Makro ausführen*
2. Makro *FormatWährung* markieren und → **OPTIONEN**
3. Feld Tastenkombination aktivieren und ⇧ (Shift) + W, **OK**

   Um das Überschreiben vordefinierter Tastenkombinationen zu vermeiden, wird die Angabe von Shift vor dem Buchstaben bei der Zuordnung der Tastenkombination empfohlen.
4. Im Dialogfeld Makro → **ABBRECHEN**

   Die Zuweisung der Tastenkombination kann bereits bei der Festlegung des Makronamens vor Beginn der Aufzeichnung erfolgen.
5. In der Budgettabelle die Felder D11:F13 auswählen und das Makro starten durch
   Ctrl + ⇧ (Shift) + W

Makro ansehen

1. In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Makro ausführen*
2. Makro *FormatWährung* markieren und → **BEARBEITEN** *oder*

   In der Symbolleiste Visual Basic klicken Sie auf die Schaltfläche *Visual Basic-Editor*

Das Fenster Microsoft Visual Basic wird eingeblendet und im Visual Basic-Editor wird ein Fenster mit dem Namen Modul1 angezeigt. Makros werden in EXCEL in Modulen abgelegt und nicht in Tabellen. Die Makros werden ebenfalls in der Arbeitsmappe gespeichert.

Der Makro beginnt mit `Sub` und endet mit `End Sub`. Alle Zeilen, die mit Apostroph beginnen sind Kommentarzeilen (grüne Schrift). Die Anweisung `Selection.NumberFormat` führt die eigentliche Arbeit aus. *`Selection`* steht für "die aktuelle Auswahl", *`Number-Format`* bezieht sich auf eine Eigenschaft oder ein Attribut der Auswahl. Beachten Sie die Verwendung des Gleichheitszeichens.

Symbolische Darstellung der Anweisung für **Eigenschaften (Attribute)**:

WOMIT.WAS TUN

Ausgewählter Bereich — Eigenschaft zuordnen *(z.B. Zahlenformat sei Währung)*

Selection.NumberFormat=" "

2. Beispiel: *Text vertikal anordnen*

1. Ordnen Sie die Fenster so an, dass sowohl die Budgettabelle als auch Modul1 zu sehen sind. A15:A20 markieren und → **MAKRO AUFZEICHNEN**
2. Makroname: *VertikalVerbinden*, Tastenkombination Shift+V, Beschreibung: Zellen vertikal verbinden und **OK**
   Beachten Sie die Eintragungen im Fenster Modul
3. → **FORMAT** → **ZELLEN** → **AUSRICHTUNG** Zellen verbinden und Ausrichtung durch Drehen des roten Pfeils nach oben auf 90 Grad setzen und **OK**
4. → **AUFZEICHNUNG BEENDEN**

   Das Makro enthält sechs verschiedene Einstellungen der Eigenschaften, die die Zellenausrichtung bestimmen. Diese entsprechen den Steuerelementen des Dialogfeldes. Das Anweisungspaar, das mit `With` beginnt und mit `EndWith` endet, wird **With-Struktur** genannt. In jeder Zeile, die mit einem Punkt beginnt, wird das Word das hinter *With* steht, vor dem Punkt eingefügt. With-Strukturen bewirken, dass ein Code einfacher zu lesen ist, da sofort erkennbar ist, dass sich alle Eigenschaften auf die aktuelle Auswahl (Selection) beziehen.
5. Wir benötigen hier nur die Eigenschaften: `Orientation` und `MergeCells`. Entfernen Sie alle unnötigen Zeilen durch Markieren und Löschen.
6. Markieren Sie in der EXCEL-Tabelle die Zellen A25:A30 und führen Sie den Makro durch Ctrl+Shift+V aus.

Empfehlung:	Aufgezeichnete Makros erscheinen übersichtlicher, wenn Sie sie von nicht benötigten, doch aufgeführten Befehlen befreien – also "ausmisten". Bei dieser Gelegenheit können Sie dann auch noch Kommentare hinzufügen.

3. Beispiel: *Gitternetzlinen entfernen / einfügen*

1. → **MAKRO AUFZEICHNEN**, Makroname: *GitternetzlinienEntfernen* **OK**
2. → **EXTRAS** → **OPTIONEN** → **ANSICHT** Gitternetzlinien entfernen und **OK**
3. → **AUFZEICHNUNG BEENDEN**
4. Betrachten Sie den Makrocode im Modul. `False` ist die Einstellung der Eigenschaft `DisplayGridlines` des aktiven Fensters und bedeutet Ausschaltung der Gitternetzlinien, dementsprechend würde `True` das Sichtbarmachen der Gitternetzlinien bedeuten.
5. Ersetzen Sie den Eintrag `False` durch `True` und führen Sie den Makro durch Drücken der Funktionstaste **F5** aus – der Cursor muss dabei im Makro *GitternetzlinienEntfernen* stehen bleiben.

   Wenn sie im Visual Basic-Editor arbeiten und das Dialogfeld *Makro* einblenden möchten, um darin ein Makro auszuwählen, klicken Sie in den Bereich unterhalb der Anweisungen, bevor Sie die Funktionstaste **F5** drücken.

Wert einer Eigenschaft mit einem Makro ändern

Statt ein Makro zum Einblenden und eines zum Ausblenden der Gitternetzlinien zu erstellen, ist es bequemer ein einziges Makro zu erstellen, das den Wert der Eigenschaft ändert. Es ist nötig, eine Variable zu definieren, welche den Wert der Eigenschaft speichert.

1. Fügen Sie im Makro *GitternetzlinienEntfernen* eine neue Zeile nach den Kommentaren ein.
2. Kopieren Sie `ActiveWindows.DisplayGridlines` in die Leerzeile und geben Sie am Anfang dieser Zeile **`neuGitter =`** ein. Der Variablenname ist also **neuGitter.**
3. Ersetzen Sie den Eintrag True durch Negation der Variablen – **`Not neuGitter`**. Das Schlüsselwort `Not` bewirkt, dass der Wert **True** in **False** und der Wert **False** in **True** geändert wird.
4. Ändern Sie den Makronamen in *GitternetzlinienSchalter* und probieren Sie es aus.

Ihr Makro sollte nun folgendes Aussehen haben:

```
Sub GitternetzlinienSchalter()
  neuGitter = ActiveWindow.DisplayGridlines
  ActiveWindow.DisplayGridlines = Not neuGitter
End Sub
```

Das Makro liest den Wert der Eigenschaft, ändert ihn mit dem Schlüsselwort `Not` in den gegenteiligen Wert und weist der Eigenschaft den neuen, invertierten Wert zu.

4. Beispiel: Formel in Wert umwandeln

1. Markieren Sie in der Tabelle BUDGET.XLS die Zelle E4. Beachten Sie die Formel in der Bearbeitungszeile: **=E3–E68**
2. → **EXTRAS** → **MAKRO** → **AUFZEICHNEN...** Makroname: *FormelInWertUmwandeln*, Tastenkombination: Shift+U
3. → **BEARBEITEN** → **KOPIEREN**
4. → **BEARBEITEN** → **INHALTE EINFÜGEN** Option **Werte** anklicken und **OK**
5. Drücken Sie auf die ESC Taste, um den Laufrahmen auszublenden
6. → **AUFZEICHNUNG BEENDEN**
7. Kontrollieren Sie den Inhalt der Zelle E4, hier soll der Wert 41918 statt der Formel stehen
8. Wechseln Sie zu Visual Basic, um das aufgezeichnete Makro anzusehen

   Da die Anweisung *PasteSpecial* mehr als 70 Zeichen enthält, erscheint am Ende der ersten Zeile das sogenannte Zeilenfortsetzungszeichen – ein Leerzeichen und ein Unterstrich _ . Die Anweisung wird in einer zweiten Zeile fortgesetzt. Um diese lange Anweisung besser lesbar zu machen, fügen Sie jedes Argument der Anweisung in eine eigene Zeile ein, so dass jede Zeile, ausser der letzten mit einem Leerzeichen und einem Unterstrich endet.

Bei den Beispielen 1 bis 3 haben wir mit den Makros eine oder mehrere **Eigenschaften** von Zellen geändert. Wenn die Makroaufzeichnung eine Anweisung erzeugt, die das **Gleichheitszeichen** = enthält, so stellt das Wort vor dem Gleichheitszeichen eine **Eigenschaft** dar z.B. bedeutet die Anweisung `ActiveWindow.DisplayGridlines = False` "False soll der Wert der Eigenschaft der Gitternetzlinien im aktiven Fenster" sein.

Im Beispiel 4 treten zwei Anweisungen `Selection` auf, die kein Gleichheitszeichen enthalten. `Copy` ist jedoch keine Eigenschaft. Aktionen wie Kopieren oder Einfügen werden **Methoden** genannt. Bei Verwendung des Befehls *Kopieren* werden Sie von Excel nicht nach zusätzlichen Informationen gefragt, daher geben Sie auch keine weiteren Informationen an die Methode `Copy` weiter.

Beim Befehl *Inhalte einfügen* (*PasteSpecial*) werden Sie in einem Dialogfeld gefragt, wie Sie diese Aktion ausführen möchten. Wenn Sie die Methode `PasteSpecial` in einem Makro einsetzen, übergeben Sie diese Zusatzinformationen an die Methode. Solche Informationen werden **Argumente** genannt. Die vier Argumente bei `PasteSpecial` entsprechen den vier Optionsgruppen des Dialogfeldes *Inhalte einfügen*. Jedes Argument besteht aus einem **Argumentnamen** z.B. `Paste` und dem **Argumentwert** z.B. `xlValues`, die durch einen Doppelpunkt und ein Gleichheitszeichen **:=** voneinander getrennt werden.

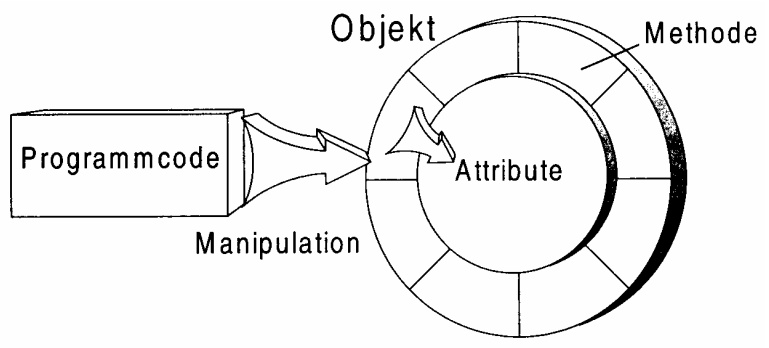
Symbolische Darstellung der Anweisung für **Methoden**:

WOMIT**.**WAS TUN

Ausgewählter Bereich	Methode angeben	*Methoden können Argumente haben, die durch Kommas getrennt werden*
	Format für Argumente:	Argumentname **:=** Argumentwert

Objektorientierte Programmierung

VBA ist wie die Sprache Visual BASIC objektorientiert. Die folgende Abbildung zeigt den objektorientierten Ansatz:



Ein Objekt repräsentiert ein Element einer Anwendung. Für EXCEL sind das beispielsweise eine Tabelle, ein Diagramm, ein Text in einer Zelle, ein Bereich etc. Für alle Elemente von EXCEL existiert ein entsprechendes Objekt in VBA. Mit Hilfe von Eigenschaften und Methoden können Sie ein einzelnes Objekt oder eine gesamte Auflistung von Objekten modifizieren.

Das folgende Beispiel soll das Konzept eines Objektes erläutern:

```
Sub test()
  Dim bereich As Range
  Set bereich = Worksheets("Tabelle1").Range("A1:B2")
  bereich.Select
  bereich.Interior.ColorIndex = 8
End Sub
```

Über VBA-Anweisungen soll ein Bereich eines Tabellenblattes markiert und daraufhin dem Hintergrund eine andere Farbe zugewiesen werden.

Objekt:

```
Dim bereich As Range
Set bereich = Worksheets("Tabelle1").Range("A1:B2")
```

Die erste Zeile des Listings deklariert ein Objekt vom Typ `Range`, das geeignet ist, einen Tabellenbereich darzustellen. Die zweite Zeile weist dem Objekt den Bereich von A1.bis B2 des Tabellenblattes mit dem Namen "Tabelle1" zu.

Methode:

Das Markieren des Bereiches erfolgt über eine spezielle Anweisung, die nur zusammen mit einem Objekt oder einem Objektnamen, der den Bereich darstellt, anwendbar ist.

```
bereich.Select
```

Das Wort `Select` ist eine Methode, d.h. ein vordefiniertes kleines Programm, dessen Aufgabe die Markierung des Bereiches ist. Methoden sind dafür zuständig mit dem Objekt bestimmte Aktionen auszuführen.

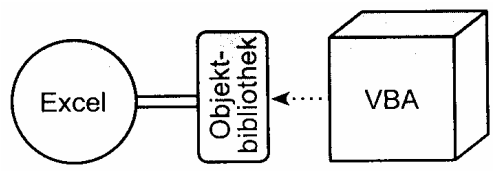
Eigenschaft:

Die Farbe des Hintergrundes ist eine Eigenschaft des Bereiches als Objekt und kann nur über das Objekt angesprochen werden.

```
bereich.Interior.ColorIndex = 8
```

Das Wort `Interior` bezeichnet das Innere oder den Hintergrund und ist als logisch untergeordnetes Objekt des Objekts `bereich` zu verstehen. Das Wort `ColorIndex` ist der Name einer der möglichen Eigenschaften oder Attribute des Objektes `Interior`. Der Wert dieser Eigenschaft wird auf 8 gesetzt – das entspricht der Farbe Zyan. Eigenschaften kontrollieren das Erscheinungsbild von Objekten, es lassen sich damit Merkmale des Objekts verändern.

Die folgende Abbildung zeigt den Zusammenhang zwischen Visual Basic und EXCEL:

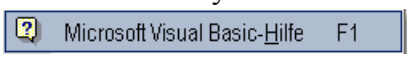


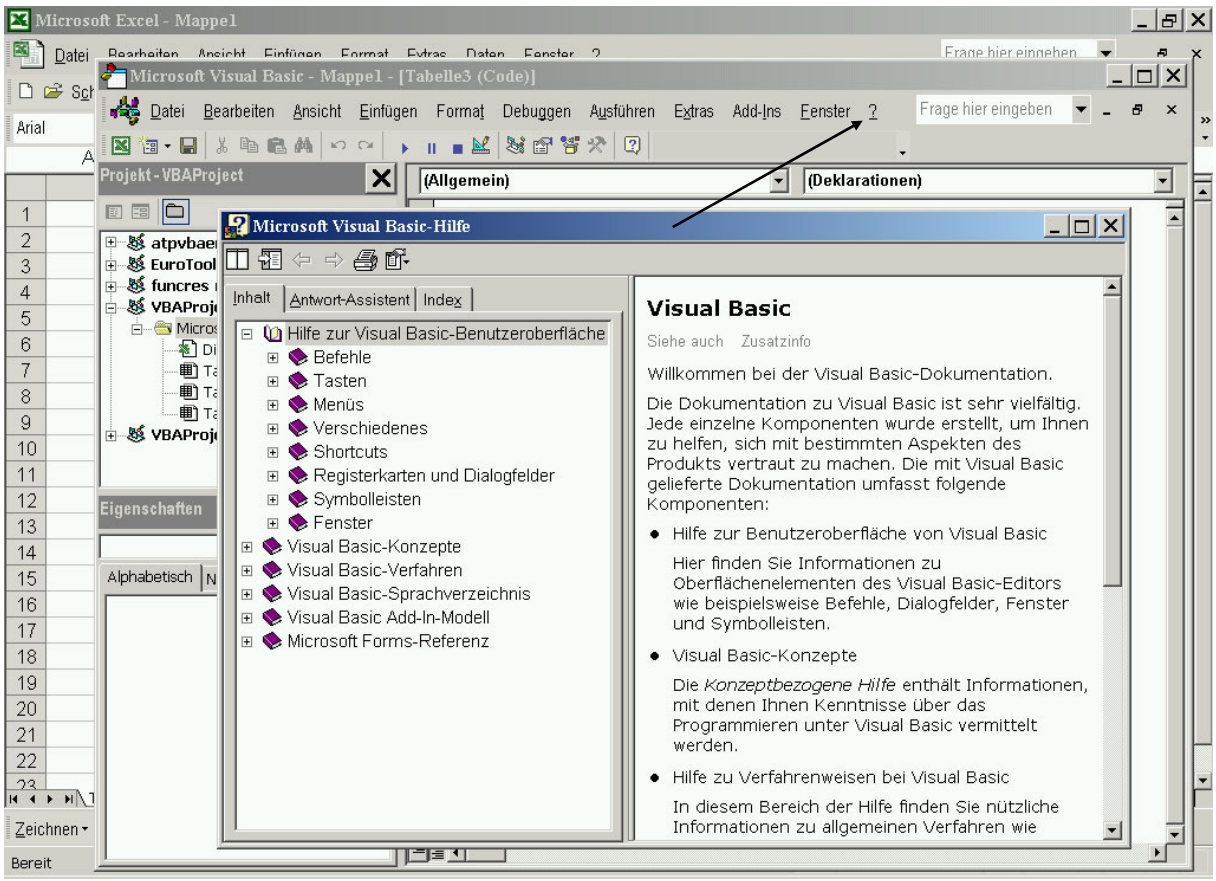
Objektbibliothek: Menge der Befehle, die die Fähigkeiten von EXCEL enthalten.

Hilfe von Visual Basic:

In der Hilfe von EXCEL finden Sie auch einiges zu VBA.

Zusätzlich können Sie vom Visual Basic Editor aus die Hilfe aktivieren:

Klicken Sie in der Standardsymbolleiste in der Visual Basic Mappe auf das Menü **?** und gehen Sie auf 



Wenn Sie bei einem aufgezeichneten Makro einen bestimmten Befehl finden, bei dem Sie nicht wissen, wozu er dient, können Sie diesen Befehl markieren und die Funktionstaste F1 drücken – damit erhalten Sie dazu einen Hilfetext!

VBA kennenlernen

VBA-Programme werden in einer speziellen Entwicklungsumgebung geschrieben: im **Visual Basic Editor**- kurz **VBE**. Dieser verfügt über ein eigenes Fenster mit eigenen Menüs und Symbolleisten, einer graphischen Projektverwaltung, Eigenschaftenfenster, einer integrierten Umgebung für die Entwicklung von benutzerdefinierten Bildschirmmasken etc.

Aufruf des VBE durch: → **EXTRAS** → **MAKROS** → **VISUAL BASIC-EDITOR**. Man erhält ein neues Fenster *Microsoft Visual Basic Mappe1* mit einer Menü- und einer Symbolleiste.

Alle Bestandteile eines Programms werden in einem Projekt zusammengefasst und mit Hilfe des **Projekt-Explorers** erfasst und verwaltet. Der Aufruf des Projekt-Explorers erfolgt durch:

→ **ANSICHT** → **PROJEKT-EXPLORER**

Man erhält ein weiteres Fenster mit der Titelleiste *Projekt VBAProject* und den Elementen des Projektes, die der aktuell geöffneten Mappe automatisch zugeordnet sind. Bei einer leeren Mappe sind dies: *Diese Arbeitsmappe* und *3 Tabellen,* die sich im Ordner Microsoft Excel Objekte befinden. Für alle gleichzeitig geöffneten Mappen wird ein getrenntes Projekt angelegt und im Projektfenster angezeigt. Die Symbolleiste zeigt drei Schaltflächen: **Code anzeigen**, **Objekt anzeigen** und **Ordner wechseln**. Es können vom Projekt-Explorer aus die Objekte ausgewählt werden, für die VBA-Anweisungen eingegeben werden. Die Eingabe der VBA-Anweisungen erfolgt in einem besonderen Code-Fenster.

Öffnen des **Code-Fensters**:

Zuerst gewünschtes Objekt im Projekt-Fenster markieren, dann → **ANSICHT** → **CODE**
Im Code-Fenster ist eine Leiste mit zwei Listenfeldern, das linke zeigt die Objekte, das rechte Listenfeld zeigt die Ereignisse, die für das jeweilige im linken Feld markierte Objekt programmiert werden können. Durch Doppelklick auf eine der *Tabellen* im Projektfenster wird das Code-Fenster der Tabelle (worksheet) geöffnet, bei Doppelklick auf *Diese Arbeitsmappe* das Codefenster der Arbeitsmappe (Workbook).

1. Beispiel *Meldung am Bildschirm beim Öffnen und Schliessen einer Mappe*

Beim Öffnen der Mappe soll am Bildschirm eine Meldung erscheinen

1. *Aufruf der Entwicklungsumgebung und Anzeige des Projekt-Explorers*

   Excel Tabelle → **EXTRAS** → **MAKRO** → **VISUAL BASIC-EDITOR**
   oder Symbol VBE aus der Visual Basic Symbolleiste *oder* Alt+F11

   Visual Basic Editor → **ANSICHT** → **PROJEKT-EXPLORER**
   oder Symbol Projekt-Explorer aus der VBE Symbolleiste *oder* Ctrl+R

   Im Projekt-Fenster *Diese Arbeitsmappe* markieren und Code-Fenster z.B. über die Schaltfläche *Code anzeigen* öffnen.

2. Im linken Listenfeld des Code-Fensters **Workbook** auswählen. Zwischen den bereits vorhandenen Zeilen `Private Sub Workbook_Open()` und `End Sub` fügen wir ein:
   `MsgBox` *"Hallo, VBA-Benutzer ! "* also beliebigen Text zwischen Anführungszeichen.

Da **`MsgBox`** ein Schlüsselwort ist, wird automatisch die komplette und richtige Schreibweise aus der Online-Hilfe eingeblendet.

Empfehlung: Schreiben Sie zunächst alle Ihre Eingaben mit Kleinbuchstaben, sind alle Angaben korrekt, so erfolgt die automatische Umwandlung in Grossbuchstaben.

3. *Ausführung des Programms:*

   Die Wirkung kann vom VBE – Fenster aus beobachtet werden

   → **AUSFÜHREN** → **SUB/USERFORM AUSFÜHREN**
   oder kurz durch Drücken der Taste **F5**
   oder über die entsprechende Schaltfläche **Sub/UserForm ausführen**

   Der Cursor muss sich dabei innerhalb der Prozedur befinden.

   Bei der Ausführung erscheint ein kleines Fenster mit der angegebenen Meldung. Mit **OK** beenden.

Beim Schliessen der Mappe soll am Bildschirm eine Meldung erscheinen.

1. Im Code-Fenster von *DieseArbeitsmappe* im rechten Listenfeld den Eintrag *BeforeClose* wählen.

2. Zwischen den bereits vorhandenen Zeilen
   `Private Sub Workbook_BeforeClose (Cancel As Boolean)` und `End Sub` fügen wir ein:
   `MsgBox "Kommen Sie bald wieder!"` oder beliebigen anderen Text.

3. Diese Prozedur kann nicht so einfach wie die erste im VBE-Fenster ausgeführt werden, da sie die Übergabe eines Parameters beim Schliessen der Mappe erwartet. Beim Schliessen der Mappe kann aber diese Meldung angesehen werden.

2. Beispiel *Meldung am Bildschirm beim Aktivieren des Tabellenblatts und in der Tabelle*

1. Aufruf der Entwicklungsumgebung und Anzeige des Projekt-Explorers mit

   → **EXTRAS** → **MAKROS** → **VISUAL BASIC-EDITOR** *oder* Symbol
   → **ANSICHT** → **PROJEKT-EXPLORER** *oder* Symbol

   Im Projekt-Fenster *Tabelle1 (Tabelle1)* markieren und Code-Fenster z.B. über die Schaltfläche *oder* mit Doppelklick öffnen.

2. Im linken Listenfeld (Objekt) **Worksheet** und im rechten Listenfeld (Prozedur) den Eintrag **Activate** auswählen.
   Zwischen den vorhandenen Zeilen `Private Sub Worksheet_Activate()` und `End Sub` fügen wir ein:

   ```
   MsgBox " Hallo VBA-Benutzer, das ist Tabelle 1!"
   ```

   Tragen Sie die entsprechenden Angaben auch bei Tabelle 2 und Tabelle 3 ein.

3. Ergänzen Sie den Code von Tabelle 1 durch die zwei weiteren Programmteile:
   (verwenden Sie bei der Eingabe die Eintragungen in den Listenfeldern Objekt und Prozedur)

```
Private Sub Worksheet_Activate()
  MsgBox "Hallo, VBA-Benutzer, das ist Tabelle 1!"
End Sub

Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Excel.Range, Cancel As Boolean)
  MsgBox "Sie haben wirklich mit der linken Maustaste Doppelklick gemacht!"
End Sub

Private Sub Worksheet_BeforeRightClick(ByVal Target As Excel.Range, Cancel As Boolean)
  MsgBox "Bravo, das war die rechte Maustaste!"
End Sub
```

4. Probieren Sie alles aus: Test der VBA- Programme zu Tabelle1
   1. Register Tabelle 2 / 3 anklicken- nachher wieder Tabelle 1
   2. Doppelklick auf beliebige Zelle in Tabelle 1
   3. Einfachklick mit der rechten Maustaste in Tabelle 1

3. Beispiel *Schliessen einer Mappe erst nach Rückfrage*

Das VBA-Programm soll in einem neuen Element- in einem **Modul**- geschrieben werden. Ein Modul ist ein Element, das nicht mit einem Tabellenblatt oder mit einer Mappe und auch nicht mit einem Formular in Verbindung gebracht werden kann.

1. Aufruf von VBE → **EINFÜGEN** → **MODUL** *oder*
   Symbol **UserForm einfügen** aufklappen und → **MODUL**
   In der Liste des Projekts im Explorer erscheint jetzt der Name für das 1. Modul: Modul1. Dieser Name könnte im Eigenschaftenfenster bei der Eigenschaft Name geändert werden.

2. Geben Sie folgenden Programmtext in den Modul ein:

```
Sub AktuelleMappeSchliessen()
  Dim meldung As String, titel As String, antwort As Byte
  titel = "Hallo, VBA - Benutzer!"
  meldung = "Soll die aktuelle Mappe wirklich geschlossen werden?"
  antwort = MsgBox(meldung, vbYesNo, titel)
  If antwort = vbYes Then
    ActiveWorkbook.Close
  Else
    MsgBox "Jetzt haben Sie NEIN gewählt!"
  End If
End Sub
```

   Bemerkung: mit → **TESTEN** → **KOMPILIEREN VON VBAPROJECT** erfolgt der Test auf Syntaxfehler

3. Ausführung des benutzerdefinierten Dialogs direkt von VBE aus:
   → **AUSFÜHREN** → **SUB/USERFORM AUSFÜHREN** *oder*
   kurz durch Drücken der Taste **F5** *oder*
   über die entsprechende Schaltfläche **Sub/UserForm ausführen**

   Testen Sie beide Möglichkeiten, geben Sie zuerst Nein danach Ja an.

Erläuterungen zum Programm

Das Programm soll, wie schon erwähnt, die aktuelle Mappe nach einer Rückfrage schliessen. Die Anweisungen im Listing bedeuten im einzelnen:

- ***Sub AktuelleMappeSchliessen():*** Das Schlüsselwort `Sub` bezeichnet in VBA eine Prozedur. Der Text *`AktuelleMappeSchliessen`* ist der Name dieser Prozedur. Die zwei Klammern () müssen eingegeben werden. Zwischen den Klammern können eventuelle Parameter deklariert werden. Im Moment wird kein Parameter gebraucht, daher bleiben sie leer.

- ***Dim meldung As String, titel As String, antwort As Byte:*** Diese Zeile deklariert 3 Variablen. Die Variablen *`meldung`* und *`titel`* sind vom Typ **String**, d.h., sie sind Zeichenketten, die Variable *`antwort`* ist numerisch und vom Typ **Byte**.

- ***titel = "Hallo, VBA-Benutzer", meldung = "Soll die aktuelle Mappe geschlossen werden?":*** Die zwei Variablen *`titel`* und *`meldung`* bekommen zwei Texte zugewiesen. Texte werden in VBA zwischen zwei " ··· " Zeichen geschrieben. Der Text der Variablen *`titel`* wird später in der Titelleiste einer Meldung erscheinen, der Text von *`meldung`* wird die Meldung selbst sein.

- ***antwort = MsgBox(meldung, vbYesNo, titel):*** Das Wort **`MsgBox`** wurde schon in anderen Beispielen dieses Kapitels verwendet, aber in einer anderen Form. `MsgBox` ist der Name einer Basic-Funktion, die auf zwei verschiedene Arten aufgerufen werden kann: Entweder in der vereinfachten Form `MsgBox` *`"beliebiger Text"`* oder wie hier angegeben. Der Funktion `MsgBox` werden in den runden Klammern drei Parameter übergeben: Der Text für die Meldung (*`meldung`*), die VBA-Konstante `vbYesNo` und der Text für die Titelleiste der Meldung (*`titel`*). Die Konstante `vbYesNo` legt fest, dass das Fenster der Meldung nur die Schaltflächen **JA** und **NEIN** besitzt. Der Anwender kann dann entweder die eine oder die andere Schaltfläche benutzen. Die Variable *`antwort`* enthält einen entsprechenden Wert.

- ***If antwort = vbYes Then ActiveWorkbook.Close:*** Hier wird ausgewertet, was der Anwender gewählt hat. Wenn er die Schaltfläche **JA** gewählt hat, wird die Mappe geschlossen. Falls sie noch nicht gespeichert war, fragt dann Excel automatisch, ob die Mappe zu speichern ist. Das Wort `vbYes` ist eine VBA-Konstante. Wenn der Benutzer "Nein" wählt, erscheint die nach dem Schlüsselwort **`Else`** angegebene Meldung und die Mappe bleibt offen. Wenn der Anwender "Ja" wählt, dann wird die Anweisung nach dem Schlüsselwort **`Then`** ausgeführt.
  Die Anweisung bezieht sich auf das Objekt *`aktuelle Mappe`* (*`ActiveWorkbook`*) und wendet auf das Objekt die Methode **`Close`** an.

Tip: Vom Code-Fenster aus kann man ganz schnell auf die Online-Hilfe zugreifen. Die Online-Hilfe-Seite, z.B. für die Funktion **`MsgBox`**, kann durch ein Klicken auf den Text `MsgBox` und das Drücken der Taste **F1** aufgerufen werden.

4. Beispiel *Erstellen eines einfachen benutzerdefinierten Dialogs mit einer Schaltfläche Die Schaltfläche soll auf einen Mausklick mit einer Meldung reagieren.*

Benutzerdefinierte Dialoge sind Fenster, die über Standardelemente wie Schaltflächen, Optionen, Listen etc. verfügen können. Dadurch ist eine direkte Einwirkung des Anwenders auf den Programmablauf möglich. Im Kapitel über Dialogfelder wird ausführlich auf die Erstellung von benutzerdefinierten Dialogfeldern eingegangen.

1. Einfügen eines Dialogs vom VBE aus: → **EINFÜGEN** → **USERFORM** *oder* Symbol 
Es erscheint ein neues Objekt mit dem Namen Userform1(Formular) in der Liste des Projekt-Explorers und ein leeres Dialogfenster.

2. Aufruf der Werkzeugsammlung**:** → **ANSICHT** → **WERKZEUGSAMMLUNG** *oder* Symbol  , wenn die Werkzeugsammlung noch nicht aktualisiert ist.
Jedes Symbol der Werkzeugsammlung ist ein Steuerelement, das in einem benutzerdefinierten Dialog eingefügt werden kann.

3. Symbol für **Befehlsschaltfläche**  in der Werkzeugsammlung anklicken. Der Cursor nimmt die Form eines Fadenkreuzes an und man zeichnet damit die Schaltfläche im **UserForm1**. Man positioniert den Cursor am besten in die linke obere Ecke, drückt die Maustaste und zieht nach unten rechts. Die Grösse der Schaltfläche kann jederzeit geändert werden.

4. *Beschriftung der Schaltfläche und des Buttons ändern:*
Aufruf des Eigenschaftenfensters → **ANSICHT** → **EIGENSCHAFTENFENSTER** *oder* Symbol  und bei der Eigenschaft Caption den Text ändern in: *Klick mich an!* Auf Wunsch können auch andere Eigenschaften geändert werden, z.B. bei markierten Button in der Eigenschaft Caption den Text ändern in *Schaltfläche*.

5. Doppelklick auf die Schaltfläche öffnet das Codefenster. Es öffnet sich ein Fenster, in diesem wird links der Name der Schaltfläche also CommandButton1 angezeigt und rechts wird der Name des Ereignisses Click angezeigt, das durch einen Mausklick ausgelöst wird.
Zwischen vorhandene Anfangs- und Endzeile geben wir zwei Anweisungen ein:

```
Unload Me
Msgbox "Ha, ich bin die neue Befehlsschaltfläche"
```

6. *Ausführung des benutzerdefinierten Dialogs direkt von VBE aus:*
→ **AUSFÜHREN** → **SUB/USERFORM AUSFÜHREN** *oder*
kurz durch Drücken der Taste **F5** *oder*
über die entsprechende Schaltfläche  **Sub/UserForm ausführen**

Die Ausführung findet in der Excel-Mappe statt. Wechseln Sie daher sofort in die EXCEL-Tabelle. Es erscheint das Fenster mit der Schaltfläche *Klick mich an*! Nach dem Klicken auf die Schaltfläche erscheint die Meldung von der Befehlsschaltfläche, die dann mit OK zu bestätigen ist. Durch Verwendung der Anweisung `Unload Me` wird der benutzerdefinierte Dialog geschlossen.

5. Beispiel *Aufzeichnen eines Makros zur Diagrammerstellung für eine Umsatztabelle; dem Makro wird in der EXCEL-Tabelle eine Schaltfläche zugewiesen.*

1. Öffnen Sie eine neue EXCEL-Tabelle und tippen Sie die Werte der angegebenen Umsatztabelle ein.

   Markieren Sie danach die Umsatztabelle – aber ohne Überschrift.

   → **MAKRO AUFZEICHEN,** Makroname**:** *DiagrammErstellen* **OK**

   Umsatztabelle:

	1.Quartal	2.Quartal	3.Quartal	4.Quartal		
1990	345	323	1230	321		Diagrammerstellung
1991	212	878	345	435		durch Anklicken der Schaltfläche
1992	645	324	675	510		
1993	33	199	456	768		Diagramm
1994	455	32	987	231		
1995	65	543	879	308		
1996	546	334	812	89		

2. Erstellen Sie mit Hilfe des Diagrammassistenten ein 3D Säulendiagramm auf einem getrennten Blatt.

3. → Schaltfläche *Makro Beenden* und → Schaltfläche *Makro Ausführen*

> Wird ein Makro nur in der Mappe verwendet, wo es gespeichert ist, kann es mit einer **Schaltfläche im Tabellenblatt** verbunden werden.

4. *Aktivieren der benötigten Symbolleiste*

   → **ANSICHT** → **SYMBOLLEISTE** → **STEUERELEMENT-TOOLBOX**

   Diese Symbolleiste enthält verschiedene Steuerelemente, die in Arbeitsblättern und Formularen eingesetzt werden können. Diese Steuerelemente werden als ActiveX-Steuerelemente bezeichnet, das sind besondere Arten von Zeichnungsobjekten, die beim Anklicken eine Aktion ausführen.

5. *Schaltfläche zeichnen*

   → STEUERELEMENT-TOOLBOX → Symbol **Befehlsschaltfläche**. Man ist im Zeichnenmodus und kann den Umriss der Schaltfläche im Arbeitsblatt zeichnen. Zeichnen Sie im Arbeitsblatt ein Rechteck von der linken oberen Ecke der Zelle G4 zur rechten unteren Ecke von H5.

   Zeichnungsobjekte werden an den Ecken einer Zelle ausgerichtet bzw. verankert, wenn Sie die Taste Alt gedrückt halten, während Sie das Objekt zeichnen. Sie können auch Alt gedrückt halten, um Objekte beim Verschieben oder bei Grössenänderungen an den Gitternetzlinien auszurichten.

6. *Schaltfläche editieren und den Namen Diagramm geben*

   Die Schaltfläche zeigt an den Rändern weisse Ziehpunkte Diese zeigen an, dass das Objekt ausgewählt ist. Wenn sie ausgewählt ist, können Sie ihre Eigenschaften ändern.

   → **STEUERELEMENT-TOOLBOX** → Symbol **Eigenschaften** anklicken

   Die Beschriftung wird von der Eigenschaft ***Caption*** geändert. Wählen Sie in unserem Beispiel den Namen ***Diagramm***. Nach Wunsch können auch Schriftart, Schriftgrösse (***Font***), Farben (***BackColor*** und ***Forecolor***) etc. angepasst werden.

7. *Verbindung zum Makro herstellen*

   Befehlsschaltfläche markieren → **STEUERELEMENT-TOOLBOX** → **CODE ANZEIGEN**

   Das Code-Fenster wird geöffnet. Es sind die Anfangs-und Endzeilen einer Prozedur zu sehen. Name des Moduls angeben, in dem sich unser Makro befindet, der Name muss mit einem Punkt beendet werden, dadurch wird die Liste der im Modul gespeicherten Prozeduren geöffnet, Name der gewünschten Prozedur wählen.

```
Private Sub CommandButton1_Click()
  Modul1.DiagrammErstellen
End Sub
```

8. Wechsel ins EXCEL-Tabellenblatt und Entwurfsmodus beenden

   → **STEUERELEMENT-TOOLBOX** → **ENTWURFSMODUS**

9. Makro ausprobieren
   Die benutzerdefinierte Befehlsschaltfläche Diagramm anklicken.

Bemerkung

Dieses durch Aufzeichnung erstellte VBA-Programm könnte umprogrammiert werden.

Vorschläge:

- nach dem Start des Programms nach dem Namen der Tabelle fragen und nach dem Bereich in der Tabelle, wo die Daten stehen, die in das Diagramm übernommen werden sollen. Falls die angegebene Tabelle nicht existiert oder die Bereichsangabe fehlerhaft ist, sollen Fehlermeldungen angezeigt werden.
- nach der Diagrammerstellung Möglichkeiten für Änderungen der Beschriftungen von Titel und / oder Achsen anbieten.

Zusammenstellung:

Das Editorfenster

Die eigentliche Bedieneroberfläche im Editor gestattet es, an mehreren Projekten parallel zu arbeiten – jedes Projekt erhält dabei sein eigenes Fenster. Das erleichtert die Arbeit, wenn Sie beispielsweise aus einem Projekt Makrocode herausnehmen und ihn woanders einsetzen wollen. Da der Editor als eigene Anwendung arbeitet, können Sie die Wirkung von Makros in EXCEL testen, ohne den Editor selbst zu verlassen.

Wie in allen Office-Programmen besteht auch im VB-Editor die Möglichkeit, Kontextmenüs mit der rechten Maustaste aufzurufen.

Der Projekt-Explorer

Vom VB-Editor rufen Sie den Projekt-Explorer auf. Dieser gibt Ihnen einen Überblick zu den bereitstehenden Arbeitsmappen, Tabellen, Diagrammen, Modulen und Formularen. Die Angaben werden hierarchisch gegliedert dargestellt. Mit einem Doppelklick auf das jeweilige Projekt öffnen Sie das zugehörige Objekt – beziehungsweise Codefenster. Stellen Sie ein Objekt beispielsweise als UserForm dar, dann können Sie sich den Code direkt über das Menü Ansicht und den Befehl Code ansehen.
Der Projekt-Explorer zeigt alle Projekte der im Moment aktiven Arbeitsmappen an. Er dient zum Anzeigen und Organisieren von Code und Objekten im Projekt.

Das Eigenschaftsfenster

Vom VB-Editor aus rufen Sie das Eigenschaftsfenster auf. Mit den Registern **ALPHABETISCH** und **NACH KATEGORIEN** können Sie nicht nur die jeweiligen Eigenschaften einsehen, sondern auch Änderungen daran vornehmen. Um eine solche Eigenschaft zu ändern, klicken Sie direkt in das Wertfeld der Eigenschaft und tragen den neuen Wert ein. Für viele Werte bietet der VB-Editor Dropdown –Felder an, die Ihnen mögliche Einstellungen anbieten.

Das Codefenster

Dieses Fenster ist – wie der Name schon sagt – Ihr Arbeitsplatz im VB-Editor. Dort geben Sie den Code ein. Beim Überarbeiten oder Entwickeln von VBA-Code stehen Ihnen folgende Hilfsmittel zu Verfügung:

- Das Fenster vervollständigt einzelne Worte, die Sie eingeben. Beginnen Sie eine Prozedur, etwa "`Sub Text()`" und drücken die *ENTER-Taste*, dann fügt der VB-Editor die Schlusszeile "`End Sub`" automatisch ein.
- Die Quickinfo hilft ihnen beim Tippen: Nachdem Sie die Anfangsbuchstaben einer Funktion oder Methode eingegeben haben, bietet ein Infofenster sofort den fertigen Text an. Er lässt sich leider nicht direkt mit der *ENTER-Taste* übernehmen, hilft aber doch, eventuelle Tippfehler zu vermeiden.
- Nach der Eingabe eines Objektnamens und dem Punkt-Bindezeichen bietet VBA Begriffe für Objekte, Methoden oder Eigenschaften in einer Dropdown-Liste an. Sie können entweder den Begriff heraussuchen oder solange weiter tippen, bis der Vorschlag passt. Um ihn dann auch wirklich anzunehmen, drücken Sie auf die *ENTER-Taste*.

VBA Codekonventionen

Um Ihnen die Arbeit zu erleichtern, kennzeichnet der VB-Editor den von Ihnen eingegebenen oder aufgezeichneten Code je nach seinen Eigenschaften in unterschiedlichen Farben:

- Befehle, die bereits in der Sprache von VBA verankert sind, erscheinen nach der Eingabe und einem Druck auf die *ENTER-Taste* in blauer Farbe. Auf diese Weise erkennen Sie gleich bei der Eingabe, ob VBA den Befehl "verstanden" hat.
- Kommentare, die Sie durch ein Apostroph einleiten, stellt der Editor in grün dar. Kommentare sollten Sie eingeben, damit Sie Ihr VBA-Programm auch später noch gut verstehen können. Durch das "*Auskommentieren*" können Sie Makros gut testen und analysieren.
- Syntaxfehler färbt der VB-Editor rot. Diese Färbung erscheint aber erst nach Abschluss der Eingabe mit der *ENTER-Taste*.
- Ein Leerzeichen und ein Unterstrich setzen eine Code-Zeile fort. Damit vermeiden Sie zu lange Zeilen und machen das Programm übersichtlicher.

VBA - Sprachelemente

Variablen

Eine Variable ist ein Ausdruck, der einen bestimmten Typ besitzt und dem ein Wert zugewiesen werden kann. Diese Zuweisung kann im Verlauf eines Programms einmal oder mehrmals stattfinden.

Zur Illustration des Begriffs Variable: man kann eine Variable wie eine "virtuelle Zelle in einer Excel-Tabelle" ansehen, die erst durch die Deklaration zum "Leben erweckt" wird, dann durch den Typ eine Formatierung erhält und zuletzt durch eine Zuweisung einen Inhalt bekommt.

Regeln für Variablennamen

- Der Name kann aus Buchstaben (auch Umlauten), Zahlen oder Unterstrichen bestehen, muss aber mit einem Buchstaben beginnen. Alle anderen Zeichen sind nicht erlaubt – also auch keine Leerzeichen.
- Verwenden Sie sinnvolle Namen – der Name darf max. 255 Zeichen lang sein!
- Name darf kein in VBA oder Excel reserviertes Wort sein. Verwenden Sie keine Namen die durch Funktionen, Anweisungen und Methoden in Visual Basic verwendet werden.

Datentypen

Die folgende Tabelle enthält die von Visual Basic unterstützten Datentypen sowie deren Speicherbedarf und Wertebereiche.

Datentyp	Speicherbedarf	Wertebereich
Byte	1 Byte	0 bis 255
Boolean	2 Bytes	True oder False
Integer	2 Bytes	-32'768 bis 32'767
Long (lange Ganzzahl)	4 Bytes	-2'147'483'648 bis 2'147'483'647
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Bytes	3.402823E38 bis –1.401298E-45 für negative Werte; 1.401298E-45 bis 3.402823E38 für positive Werte
Double (Gleitkommazahl mit doppelter Genauigkeit)	8 Bytes	-1.79769313486232E308 bis –4.94065645841247E-324 für negative Werte; 4.94065645841247E-324 bis 1.79769313486232E308 für positive Werte
Currency (skalierte Ganzzahl)	8 Bytes	-922'337'203'685'477.5808 bis 922'337'203'685'477.5807
Decimal	14 Bytes	+/-79'228'162'514'264'337'593'543'950'335 ohne Dezimalzeichen; +/-7.9228162514264337593543950335 mit 28 Nachkommastellen; die kleinste Zahl ungleich Null ist +/-0.0000000000000000000000000001
Date	8 Bytes	1. Januar 100 bis 31. Dezember 9999
Object	4 Bytes	Beliebiger Verweis auf ein Objekt vom Typ Object

Datentyp	Speicherbedarf	Wertebereich
String (variable Länge)	10 Bytes plus Zeichenfolgenlänge	0 bis ca. 2 Milliarden
String (feste Länge)	Zeichenfolgenlänge	1 bis ca. 65.400
Variant (mit Zahlen)	16 Bytes	Numerische Werte im Bereich des Datentyps Double
Variant (mit Zeichen)	22 Bytes plus Zeichenfolgenlänge	Wie bei String mit variabler Länge
Benutzerdefiniert (mit Type)	Zahl ist von Elementen abhängig	Der Bereich für jedes Element entspricht dem Bereich des zugehörigen Datentyps

Anmerkung:

Benutzerdefinierte Datentypen werden zur Gruppierung von zusammengehörenden Elementen verwendet und können ein oder mehrere Elemente eines beliebigen Datentyps enthalten. Ein klassischer Anwendungsbereich sind Datensätze einer Datenbank.

Auch in VBA ist die Verwendung von Datenfeldern eines beliebigen Datentyps (Arrays) wie in anderen Programmiersprachen möglich. Zusätzlich zu dem von den Elementen benötigten Speicherplatz werden 20 Bytes für die Verwaltung benötigt.

Achtung Rundungsfehler:

VBA rundet bei der Berechnung von Variablen vom Typ Byte, Integer oder Long eigenmächtig. Zum Beispiel:

Die Zahl 1.5 wird richtig auf 2 aufgerundet, die Zahl 2.5 wird falsch auf 2 abgerundet.
Die Zahl 3.5 wird richtig auf 4 aufgerundet, die Zahl 4.5 wird falsch auf 4 abgerundet etc.

Damit keine falschen Ergebnisse entstehen, sollten Sie ganzzahligen Variablen nie eine Dezimalzahl zuweisen oder diese Zahl zuerst runden und dann zuweisen.

Deklaration von Variablen

Die Deklaration von Variablen kann in VBA auf verschiedene Arten vorgenommen werden:

Implizite Deklaration:

man schreibt die Variable im Programm an der Stelle wo sie gebraucht wird, ohne sich um Einzelheiten des Variablentyps zu kümmern. Diese Variablen bekommen automatisch den Typ Variant zugewiesen. Eine Variant-Variable passt ihren tatsächlichen Typ an ihren aktuellen Inhalt an, benötigt aber mehr Speicherplatz als die meisten anderen Variablen.

Problem: bei Schreibfehlern könnten ungewollt neue Variablen deklariert werden. Es kann auch passieren, dass Sie zweimal den gleichen Namen für verschiedene Variable verwenden.

Bei grösseren Programmen unbedingt Deklaration verwenden!

Explizite Deklaration

Normalerweise verwendet man die **Dim-Anweisung** zur Deklaration.
Es wird empfohlen, alle Variablen mit Datentyp in der Dim –Anweisung am Anfang des Programmteils anzugeben.

Das Wort `Dim` ist eine Abkürzung von Dimension und hat damit zu tun, dass durch diese Deklaration dem Computer mitgeteilt wird, wie viel Platz für die Variablen zu reservieren ist.

Variablen können als mit einem der folgenden Datentypen deklariert werden:
Boolean, Byte, Integer, Long, Currency, Single, Double, Date, String (für Zeichenfolgen variabler Länge), String * Länge (für Zeichenfolgen fester Länge), Object oder Variant. Wenn Sie keinen Datentyp angeben, wird der Datentyp Variant standardmässig zugewiesen.

Man kann mehrere Variablen in einer Anweisung deklarieren. Wenn sie einen Datentyp angeben möchten, so müssen Sie den Datentyp für jede Variable angeben. In folgender Anweisung werden alle Variablen vom Typ Integer deklariert:

```
Dim x As Integer, y As Integer, z As Integer
```

Bei Verwendung der Deklaration:

```
Dim x, y, z As Integer
```

Wird nur `x` als Integer definiert, `y` und `z` erhalten Typ Variant.

Wenn Sie in Visual Basic die Verwendung der expliziten Deklarationen erzwingen möchten, müssen Sie die **`Option Explicit`** – Anweisung in dem Modul vor allen anderen Prozeduren verwenden. Enthält ein Modul diese Anweisung, so erfolgt zur Kompilierungszeit eine Fehlermeldung, wenn Visual Basic auf einen Variablennamen trifft, der nicht deklariert oder falsch geschrieben wurde.

Lokale und globale Variable:

- **Lokale Variable** sind Variable, die nur in den jeweiligen Prozeduren oder Funktionen verwendet werden können, in denen sie deklariert sind.
- **Globale Variable**, sind Variable, die am Anfang eines Moduls stehen. Diese können in allen Prozeduren und Funktionen des Moduls verwendet werden.

Konstanten

Konstante sind Werte, die sich während des gesamten Programmablaufs nicht ändern. Durch die Deklaration einer Konstanten können Sie einem konstanten Wert einen aussagefähigen Namen zuweisen.

Wenn man z.B. Berechnungen von Mehrwertsteuerbeträgen ausführt, ist der Mehrwertsteuersatz ein konstanter Wert für alle möglichen Berechnungen; daher ist es überflüssig die Mehrwertsteuer als Variable zu definieren. Verwenden Sie die Const-Anweisung, um folgende Konstante zu deklarieren und deren Wert festzulegen:

```
Const mehrwertsteuer As Single = 0.065
```

Im VBA-Programm wird dann der Name *mehrwertsteuer* verwendet. Bei Änderung des Mehrwertsteuersatzes muss dann im Programm nur diese Konstante an einer Stelle bei der Deklaration geändert werden.

Konstanten können mit einem der folgenden Datentypen deklariert werden: Boolean, Byte, Integer, Long, Currency, Single, Double, Date, String oder Variant. Da Sie den Wert einer Konstanten bereits kennen, können Sie den Datentyp in einer Const-Anweisung angeben.

Sie können mehrere Konstanten in einer Anweisung deklarieren. Wenn Sie einen Datentyp angeben, müssen Sie ihn für jede Konstante angeben.

In der folgenden Anweisung werden die Konstanten conAlter und conEinkommen als Integer bzw. Currency deklariert:

```
Const conAlter As Integer = 34, conEinkommen As Currency = 35000
```

Neben diesen **benutzerdefinierte Konstanten** stehen in VBA auch eine grosse Zahl von **vordefinierten Konstanten** zur Verfügung, die nicht deklariert werden müssen.

Während der Eingabe des Programmcodes wird von VBE an allen Stellen, an denen die Verwendung einer integrierten Konstanten sinnvoll ist, eine Liste der Konstanten angeboten. Siehe z.B. die Konstanten für die Funktion MsgBox.

Sie können eine Konstante innerhalb einer Prozedur oder zu Beginn eines Moduls im Deklarationsabschnitt deklarieren. Konstanten auf Modulebene sind standardmässig privat. Um eine öffentliche Konstante auf Modulebene zu deklarieren, müssen Sie der Const-Anweisung das Schlüsselwort **Public** voranstellen. Sie deklarieren eine private Konstante explizit, indem Sie der Const-Anweisung das Schlüsselwort **Private** voranstellen.

In dem folgenden Beispiel wird die Public-Konstante conAlter als Integer deklariert und ihr der Wert 34 zugewiesen: `Public Const conAlter As Integer = 34`

Gültigkeitsbereich von Deklarationen

Mit Gültigkeitsbereich wird die Verfügbarkeit von Variablen, Konstanten oder Prozeduren zur Verwendung durch andere Prozeduren bezeichnet.

Der Gültigkeitsbereich einer Deklaration bezeichnet die Stellen eines Programms, in denen eine Deklaration wirksam ist. Ausserhalb des Gültigkeitsbereichs ist eine Variable, Konstante oder Prozedur unbekannt. Je nachdem, ob eine Deklaration innerhalb einer Prozedur oder auf Modulebene vorgenommen wird, wird die Deklaration als lokal oder global bezeichnet.

Insgesamt gibt es drei verschiedene Ebenen für die Gültigkeitsbereiche:

- lokale Gültigkeit
- Gültigkeit auf Modulebene
- Öffentliche Gültigkeit

Lokale Gültigkeit:

Alle Variablen und Konstanten, die in einer Prozedur oder Funktion deklariert sind, besitzen nur eine lokale Gültigkeit d.h. ihr Wert ist nur dort verfügbar.

Beispiel:

```
'Lokale Gültigkeit von Variablen und Konstanten
'Prozeduren alle in einem Modul
Sub schweiz_1()
  Dim endpreis As Single, nettopreis As Single
  Const mwst_ch As Single = 0.065
  nettopreis = 100
  endpreis = nettopreis + nettopreis * mwst_ch
  MsgBox mwst_ch
  MsgBox endpreis
End Sub

'Konstante mwst_ch ist in der folgenden Prozedur nicht verwendbar
'der Endpreis ist 0
Sub schweiz_2()
  MsgBox mwst_ch
  MsgBox endpreis
End Sub

Sub deutschland_1()
  Dim endpreis As Single, nettopreis As Single
  Const mwst_d As Single = 0.15
  nettopreis = 100
  endpreis = nettopreis + nettopreis * mwst_d
  MsgBox mwst_d
  MsgBox endpreis
End Sub
```

Die Prozedur ***schweiz_2*** gibt zwei Meldungen am Bildschirm aus, aber beide sind leer – es erscheint nur **OK**. Die zweite Prozedur kann nicht auf die Werte zugreifen, die in der ersten Prozedur zugewiesen wurden.
Die Prozedur ***deutschland_1*** ist eine Kopie von *schweiz_1* mit angepasstem Mehrwertsteuersatz.

Gültigkeit auf Modulebene

Variablen und Konstanten, die für alle Prozeduren und Funktionen eines Moduls gültig sein sollen, müssen auf Modulebene deklariert werden. Einer Variablen kann dann auch in allen Prozeduren und Funktionen des Moduls ein Wert zugewiesen werden.

Beispiel:

```
'Deklaration auf Modulebene
'alle Prozeduren in einem Modul
Dim endpreis As Single, nettopreis As Single
Const mwst_ch As Single = 0.2

Sub schweiz_1()
  nettopreis = 100
  endpreis = nettopreis + nettopreis * mwst_ch
  Selection.Style = "Percent"
  MsgBox Format(mwst_ch, "percent")
  MsgBox endpreis
End Sub

Sub schweiz_2()
  MsgBox Format(mwst_ch, "0.0%")
  MsgBox "Endpreis = " & endpreis
End Sub
```

Auch die Prozedur 2 zeigt jetzt in den beiden Meldungsfenstern die richtigen Werte - vorausgesetzt es wurde **zuerst** *schweiz1()* und danach *schweiz2()* ausgeführt.

Die Deklaration auf Modulebene wird auch als Deklaration auf **privater Modulebene** bezeichnet:

Der Gültigkeitsbereich der Deklaration ist nur im privaten Bereich des Moduls wirksam d.h. nur für die Prozeduren des Moduls selbst.

Die Schreibweise:

```
Dim endpreis As Single, nettopreis As Single
Const mwst_ch As Single = 0.20
```

ist mit folgender Schreibweise gleichwertig:

```
Private endpreis As Single, nettopreis As Single
Private Const mwst_ch As Single = 0.20
```

Öffentliche Gültigkeit

Variablen und Konstanten, die für alle Prozeduren und Funktionen aller Module des Projekts gültig sein sollen, müssen auf Modulebene mit dem vorangestellten Schlüsselwort **Public** deklariert werden. Eine derart definierte Konstante kann in allen Prozeduren und Funktionen des Projektes verwendet werden, eine als öffentlich deklarierte Variable kann in allen Prozeduren und Funktionen des Projekts einen Wert entgegennehmen.

Beispiel:

Das vorherige Beispiel wird um ein zweites Modul erweitert. Wenn man in das bestehende Projekt ein neues Modul einfügt, so erhält dieses automatisch den Namen Modul2. Im **Modul1** erfolgt die **Berechnung**, im **Modul2** die **Ausgabe**. Die Prozedurnamen wurden entsprechend geändert.

```
'Deklaration auf Modulebene im Modul1
Public endpreis As Single, nettopreis As Single
Public Const mwst_ch As Single = 0.20

'Prozedur im Modul1
Sub berechnen()
  nettopreis = 100.00
  endpreis = nettopreis + nettopreis * mwst_ch
End Sub

'Prozedur im Modul2
Sub ausgabe()
  MsgBox Format(mwst_ch, "0.0%")
  MsgBox "Endpreis = " & endpreis
End Sub
```

Die Ergebnisse sind bei diesem Beispiel die gleichen wie vorher.

Operatoren

Ein Operator ist ein Symbol, das zur Durchführung mathematischer oder logischer Operationen, für die Verknüpfung oder für den Vergleich mehrerer Elemente verwendet wird.

Operatoren	*Beschreibung*
Arithmetische Operatoren	Operatoren zum Durchführen mathematischer Berechnungen
+	Addieren
–	Subtrahieren
*	Multiplizieren
/	Dividieren, mit Fliesskomma-Zahl als Ergebnis
\	Dividieren, mit ganzer Zahl als Ergebnis
^	Potenzieren (z.B. x^2)
Mod	Rest einer ganzzahligen Division
Vergleichsoperatoren	Operatoren zum Durchführen von Vergleichen
=	gleich
<	kleiner als
<=	kleiner gleich
>	grösser als
>=	grösser gleich
<>	ungleich
Is	Vergleich von Objekten
Like	Vergleich von Zeichenketten
Logische Operatoren	Operatoren zum Durchführen logischer Operationen
AND	Logisches Und
OR	Logisches Oder
NOT	Logisches Nicht
XOR	Exklusives (entweder) Oder
EQV	Logisches Äquivalenz
IMP	Logisches Implikation
Verkettungsoperatoren	Operatoren zum Aneinanderhängen von Zeichenfolgen
&	verkettet zwei Zeichenfolgen
+	kann wie "&" eingesetzt werden

Ausdrücke

Variablen und Konstante werden mit Hilfe von Operatoren zu Ausdrücken verkettet. Wenn ein Ausdruck mehrere Operationen enthält, werden die einzelnen Teilausdrücke in einer bestimmten Rangfolge ausgewertet und aufgelöst, die als Operatorvorrang bezeichnet wird.

Wenn Ausdrücke Operatoren aus mehreren Kategorien enthalten, gelten folgende Prioritätsregeln:

- logische Operatoren *höchste Priorität*
- Vergleichsoperatoren
- arithmetische Operatoren *niedrigste Priorität*

Für die arithmetischen Operatoren gilt folgende Rangfolge:

- Potenzierung (^) *höchste Priorität*
- Negation (–), Multiplikation und Division (*, /)
- Ganzzahldivision (\)
- Restwert (Mod)
- Addition und Subtraktion (+, –) *niedrigste Priorität*

Multiplikation und Division innerhalb eines Ausdrucks sind gleichrangig und werden von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Dasselbe gilt für Additionen und Subtraktionen, die zusammen in einem Ausdruck auftreten. Mit Klammern kann diese Rangfolge ausser Kraft gesetzt werden, damit bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden. In Klammern gesetzte Operationen haben grundsätzlich Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren. Bei den arithmetischen Operatoren gelten also die Regeln der Schulmathematik: z.B. Punktrechnung vor Strichrechnung.

Die Vergleichsoperatoren haben alle dieselbe Priorität und werden daher von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Sehr häufig werden arithmetische Ausdrücke durch Vergleichsoperatoren in Relation gesetzt werden.

Logische Operatoren werten Ausdrücke aus und liefern als Ergebnis einen Wahrheitswert "Wahr" oder "Falsch". Die logischen Operatoren sind für die Auswertung von Bedingungen wichtig (siehe Entscheidungen und Schleifen!)

Für die logischen Operatoren gilt folgende Rangfolge:

- Not *höchste Priorität*
- And
- Or *niedrigste Priorität*

Wertzuweisungen

Zuweisungsanweisungen weisen einen Wert oder einen Ausdruck einer Variablen oder Konstanten zu. Zuweisungsanweisungen enthalten immer ein Gleichheitszeichen (=).

VARIABLE = AUSDRUCK

Das folgende Beispiel weist den Rückgabewert der InputBox-Funktion der Variablen *IhrName* zu.

```
Sub Frage()
  Dim IhrName As String
  IhrName = InputBox("Wie heissen Sie?")
  MsgBox "Ihr Name lautet " & IhrName
End Sub
```

Anweisungen, die Werte von Eigenschaften festlegen, sind ebenfalls Zuweisungsanweisungen. Das folgende Beispiel legt die **Bold**-Eigenschaft des **Font**-Objekts für die aktive Zelle fest:

```
ActiveCell.Font.Bold = True
```

Die Set-Anweisung wird verwendet, um ein Objekt einer Variablen zuzuweisen, die als Objekt deklariert wurde. Das Schlüsselwort **`Set`** ist erforderlich. Im folgenden Beispiel weist die Set-Anweisung der Objektvariablen Zelle1 einen Bereich (hier die Zelle A1) in Tabelle1 zu, diese wird fett und kursiv formatiert.

```
Sub Formatieren()
Dim Zelle1 As Range
  Set Zelle1 = Worksheets("Tabelle1").Range("A1")
  With Zelle1.Font
    .Bold = True
    .Italic = True
  End With
End Sub
```

Bemerkungen:

Fortsetzen einer Anweisung über mehrere Zeilen

Eine Anweisung belegt normalerweise eine Zeile. Sie können aber auch eine Anweisung in der nächsten Zeile fortsetzen, indem Sie ein Zeilenfortsetzungszeichen (ein Leerzeichen und ein Unterstrich _) verwenden.

Mehrere Anweisung in einer Zeile

Sie können durch Doppelpunkt getrennt mehrere Anweisungen in einer Zeile positionieren.

Hinzufügen von Kommentaren

Kommentare können eine Prozedur oder eine bestimmte Anweisung erklären. Visual Basic ignoriert Kommentare beim Ausführen der Prozeduren. Kommentarzeilen beginnen mit einem Apostroph (') oder mit `Rem` gefolgt von einem Leerzeichen und können überall in einer Prozedur eingefügt werden. Geben Sie einen Apostroph gefolgt von einem Kommentar nach der Anweisung an, um diesen Kommentar auf die gleiche Zeile zu setzen wie die Anweisung. Standardmässig werden Kommentare als grüner Text angezeigt.

Überprüfen von Syntax-Fehlern

Wenn Sie die *ENTER-Taste* nach Eingabe einer Code-Zeile drücken und die Zeile in rot angezeigt wird (oder eine Fehlermeldung angezeigt wird), müssen Sie den Fehler in Ihrer Anweisung ermitteln und korrigieren.

Programmstrukturen

Entscheidungen

Mit bedingten Anweisungen können innerhalb des Programms Entscheidungen über den weiteren Programmablauf getroffen werden. In Abhängigkeit von Bedingungen wird von mehreren Möglichkeiten eine (oder auch keine) zur Ausführung ausgewählt.

Wählen der zu verwendenden bedingten Anweisung

- **If ··· Then ··· Else:** Verzweigen, wenn eine Bedingung **True** oder **False** ist.
- **Select Case:** Auswählen einer Verzweigung aus einer Gruppe von Bedingungen.

Einfache Verzweigung mit maximal zwei möglichen Wegen

Syntax

```
If Bedingung Then
  Anweisungen
[ElseIf Bedingung-n Then
  [elseifAnweisungen]]
[Else
  [elseAnweisungen]]
End If
```

Der `Then` – Zweig wird ausgeführt, wenn die Bedingung wahr ist, der `Else` – Zweig wird ausgeführt, wenn die Bedingung nicht wahr ist.

Beispiele:

```
Sub TestWenn1()
  Dim txt As String
  txt = "Hund"
  If txt = "Katze" Then
    MsgBox "Miau!"
  Else
    MsgBox "Wau!"
  End If
End Sub
```

Falls hinter `Then` und `Else` jeweils nur eine Anweisung folgt, kann die gesamte Anweisung in einer Zeile geschrieben werden, sie muss aber dann **ohne** `End If` programmiert werden:

```
If txt = "Katze" Then MsgBox "Miau!" Else MsgBox "Wau!"
```

Empfehlung: Struktur mit `End If` verwenden

```
Sub TestWenn2()
  Dim txt As String
  txt = "Hund"
  If txt = "Hund" Then
    MsgBox "Wau!"
  End If
End Sub
```

Da hier nach `Then` nur eine Anweisung folgt, könnte auch diese Anweisung in einer Zeile geschrieben werden und zwar **ohne** `End If`: `If txt = "Hund" Then MsgBox "Wau!"`

> Wenn der `Then` - Zweig aus mehreren Anweisungen besteht, muss **tückischerweise** nach `Then` ein Zeilenumbruch folgen und die Anweisung muss mit `End If` terminiert werden.

Die Bedingungen können die verschiedenen Vergleichsoperatoren und auch logische Operatoren enthalten. Beachten sie dabei die Regeln über Operatorvorrang.

Ausserdem können `If` - Anweisungen auch verschachtelt werden:

```
Sub TestWenn3()
  Dim txt As String, msg As String
  txt = "Katze"
  If txt = "Hund" Then
    msg = "Wau!"
  ElseIf txt = "Kuh" Then
        msg = "Muh!"
      Else
        msg = "Miau!"
  End If
  MsgBox msg
End Sub
```

Mehrfache Verzweigung

Verschiedene Probleme erfordern die Unterscheidung von mehr als zwei Fällen.
Statt komplizierten Verschachtelungen von `If` - Anweisungen verwendet man besser die mehrfache Verzweigung mit Hilfe der `Select – Case` Anweisung. Dadurch wird das Programm leichter verständlich.

Syntax

```
Select Case Testausdruck
  [Case Ausdrucksliste-n
    [Anweisung-n]] ...
  [Case Else
    [elseAnw]]
End Select
```

```
Sub TestMehrfach1()
  Dim txt As String, msg As String
  txt = "Igel"
  Select Case txt
    Case "Hund"
      msg = "Wau!"
    Case "Katze"
      msg = "Miau!"
    Case "Kuh"
      msg = "Muh!"
    Case Else
      msg = "Keine Ahnung!"
  End Select
  MsgBox msg
End Sub
```

Der letzte Zweig **`Case Else`** kann auch weggelassen werden. Wenn aber die Variable ***txt*** einen anderen als die Werte ***Hund, Katze, Kuh*** hätte würde dies zu einem Fehler und zu Programmabbruch führen.

Im folgenden Beispiel überprüft die `Select Case` – Anweisung den Inhalt der Variablen Leistung. Beachten Sie, dass die `Case` – Anweisung auch mehrere Werte, einen Bereich von Werten oder eine Kombination von Werten und Vergleichsoperatoren enthalten kann.

Testen Sie das Beispiel, indem Sie verschiedene Werte für Leistung eintragen:

```
Sub einkommenBonus()
  Einkommen = 1000
  Leistung = 5
  Select Case Leistung
    Case 1
      Bonus = Einkommen * 0.1
    Case 2, 3
      Bonus = Einkommen * 0.09
    Case 4 To 6
      Bonus = Einkommen * 0.07
    Case Is > 8
      Bonus = 100
    Case Else
      Bonus = 0
  End Select
  MsgBox Bonus
End Sub
```

Schleifen

Schleifen ermöglichen die wiederholte Ausführung einer Gruppe von Anweisungen.

In Visual Basic steht eine verwirrend grosse Anzahl von Schleifen zur Verfügung. Einige Schleifen wiederholen Anweisungen, bis eine Bedingung dem Wert False entspricht, andere, bis eine Bedingung dem Wert True entspricht. Der Test der Bedingung kann am Anfang oder am Ende der Schleife erfolgen. Es gibt ausserdem Schleifen, die Anweisungen für jedes Objekt in einer Auflistung oder mit einer festgelegten Anzahl an Wiederholungen ausführen.

Überblick:

- Do...Loop: Ausführen der Schleife, *wenn* bzw. *solange bis* eine Bedingung dem Wert *True* bzw. *False* entspricht. Es gibt vier Versionen von Do...Loop's.
- For...Next: Verwenden eines Zählers, um Anweisungen mit einer festgelegten Anzahl an Wiederholungen auszuführen.
- For Each...Next: Wiederholen einer Gruppe von Anweisungen für jedes Objekt in einer Auflistung.
- While...Wend: Solange eine Bedingung erfüllt ist, wird die Schleife ausgeführt.

Wir illustrieren alle möglichen Schleifenarten durch kleine Beispiele, damit Sie die verschiedenen Arten kennenlernen und verstehen können. Wenn Sie selbst programmieren, können sie natürlich Ihre eigene Auswahl treffen.

Do...Loop - Anweisung

Syntax

```
Do [{While | Until} Bedingung]
  [Anweisungen]
  [Exit Do]
  [Anweisungen]
Loop
```

```
Do
  [Anweisungen]
  [Exit Do]
  [Anweisungen]
Loop [{While | Until} Bedingung]
```

DoLoopUntil ***REPEAT*-Schleife in *Pascal***

Die Bedingung wird am Ende der Schleife überprüft, d.h. die Schleife wird mindestens einmal ausgeführt. Die Ausführung der Schleife wird solange fortgesetzt, bis die Bedingung wahr ist:

```
Sub TestDoLoopUntil()
  Dim i As Byte, msg As String
  i = 1
  Do
    i = i + 1
    msg = msg & i & ", "
  Loop Until i = 10
  MsgBox msg
End Sub
```

DoUntil

Die Bedingung wird am Anfang der Schleife überprüft, d.h. es kann sein, dass die Schleife überhaupt nicht ausgeführt wird. Die Ausführung der Schleife wird solange fortgesetzt, bis die Bedingung wahr ist:

```
Sub TestDoUntil()
  Dim i As Byte, msg As String
  i = 1
  Do Until i = 10
    i = i + 1
    msg = msg & i & ", "
  Loop
  MsgBox msg
End Sub
```

DoLoopWhile

Die Bedingung wird am Ende der Schleife überprüft, d.h. die Schleife wird mindestens einmal ausgeführt. Die Ausführung der Schleife wird solange fortgesetzt, bis die Bedingung falsch ist:

```
Sub TestDoLoopWhile()
  Dim i As Byte, msg As String
  i = 1
  Do
    i = i + 1
    msg = msg & i & ", "
  Loop While i < 10
  MsgBox msg
End Sub
```

DoWhile

Die Bedingung wird am Anfang der Schleife überprüft, d.h. es kann sein, dass die Schleife überhaupt nicht ausgeführt wird.. Die Ausführung der Schleife wird solange fortgesetzt, bis die Bedingung falsch ist:

```
Sub TestDoWhile()
  Dim i As Byte, msg As String
  i = 1
  Do While i < 10
    i = i + 1
    msg = msg & i & ", "
  Loop
  MsgBox msg
End Sub
```

For...Next – Anweisung ***FOR*-Schleife in *Pascal***

Syntax

```
For Zähler = Anfang To Ende [Step Schritt]
  [Anweisungen]
  [Exit For]
  [Anweisungen]
Next [Zähler]
```

Zähler	Erforderlich. Numerische Variable, die als Schleifenzähler dient. Eine boolesche Variable oder ein Element eines Datenfeldes ist nicht zulässig.
Anfang	Erforderlich. Startwert von Zähler.
Ende	Erforderlich. Endwert von Zähler.
Schritt	Optional. Betrag, um den Zähler bei jedem Schleifendurchlauf verändert wird. Falls kein Wert angegeben wird, ist die Voreinstellung für Schritt **eins**.
Anweisungen	Optional. Eine oder mehrere Anweisungen zwischen For und Next, die so oft wie angegeben ausgeführt werden.

Bemerkungen Das Argument Schritt ist entweder positiv oder negativ. Der Wert des Arguments Schritt legt die Schleifenausführung folgendermassen fest:

Wert	Schleifenausführung, wenn
Positiv oder 0	Zähler <= Ende
Negativ	Zähler >= Ende

Nachdem alle Anweisungen in der Schleife ausgeführt wurden, addiert das Programm Schritt zum Wert von Zähler hinzu. Die Anweisungen in der Schleife werden dann entweder erneut ausgeführt oder das Programm beendet die Schleife und setzt die Ausführung mit der auf die Next-Anweisung folgenden Anweisung fort.

Die For-Anweisung wird immer bis zu oberen Grenze ausgeführt, ausser wenn die Anweisung `Exit For` verwendet wird.

For - Beispiele

Im Beispiel `ZweierSumme` wird die Zählervariable `j` bei jedem Schleifendurchlauf um 2 erhöht. Wenn die Schleife beendet wird, erhält man die Summe von 2, 4, 6, 8 und 10.

```
Sub ZweierSumme()
  For j = 2 To 10 Step 2
    Summe = Summe + j
  Next j
  MsgBox "Die Summe beträgt " & Summe
End Sub
```

Im Beispiel `NeueZweierSumme` wird die Zählervariable `meineZahl` bei jedem Schleifendurchlauf um 2 verringert. Verwenden Sie einen negativen Step-Wert, um die Zählervariable zu verringern und geben Sie einen Endwert an, der geringer als der Anfangswert ist. Wenn die Schleife beendet wird, erhält man die Summe von 16, 14, 12, 10, 8, 6, 4 und 2.

```
Sub NeueZweierSumme()
  For meineZahl = 16 To 2 Step -2
    Summe = Summe + meineZahl
  Next meineZahl
  MsgBox "Die Summe beträgt " & Summe
End Sub
```

Das Beispiel `TestFor1` zeigt die Möglichkeit, die Buchstaben des Alphabets anzuzeigen:

```
Sub TestFor1()
  Dim i As Byte, txt As String
  For i = 1 To 26
    'Die Funktion CHR() gibt das Zeichen der ASCII-Tabelle zurück,
    'das ihrem Parameter entspricht.
    txt = txt & Chr(i + 64) 'Grossbuchstaben
  Next
  txt = txt & Chr(13) & Chr(10)          'bewirkt Zeilenvorschub
  For i = 1 To 26
    txt = txt & Chr(i + 96) 'Kleinbuchstaben
  Next
  MsgBox txt
End Sub
```

Das Beispiel `TestFor2` zeigt einige Möglichkeiten der Textbearbeitung. Es wird die Länge eines Satzes ermittelt und der Satz wird nach einem bestimmten Zeichen durchsucht:

```
Sub TestFor2()
  Dim n As Byte, npos As Byte, i As Byte
  Dim txt As String, msg As String
  'Ein italienischer Zungenbrecher soll nach dem Zeichen ","
  'durchgesucht werden
  txt = "Chi troppo in alto sal, cade sovente precipitevolissimevolmente!"
  'Länge der Zeichenkette ermitteln
  n = Len(txt)
  'Die Zeichenkette wird zeichenweise untersucht
  For i = 1 To n
    If Mid$(txt, i, 1) = "," Then
      npos = i
      Exit For
    End If
  Next
msg = "Kommaposition im String: " & npos
  MsgBox msg
End Sub
```

```
'Eine Alternative zu der obigen FOR Schleife ist die
'Funktion InStr → npos = InStr(1, txt, ",")
```

Sie können **For ⋯ Next** - Schleifen auch verschachteln, indem Sie eine For ⋯ Next - Schleife innerhalb einer anderen verwenden. Das Argument Zähler muss für jede Schleife einen eindeutigen Variablennamen erhalten. Beachten Sie folgende Konstruktion:

```
For I = 1 To 10
  For J = 1 To 10
    For K = 1 To 10
      ...
    Next K
  Next J
Next I
```

Das Beispiel ***Zahlenlotto_ch*** "6 aus 45" zeigt die Verschachtelung der Schleifen `DO ⋯ Loop Until` und `For ⋯ Next` und den Gebrauch vom Datentyp **Datenfeld** (**Array**). Die Zahlen werden über Zufallszahlen ermittelt. Jede Zahl darf natürlich nur einmal vorkommen. Wir verwenden zur Kontrolle von gleichen Zahlen die logische Variable *gefunden (*`Boolean`*)* und die `Exit For` -Anweisung.

```
Sub Zahlenlotto_ch()
  Dim zahlen(5) As Integer, i As Integer, j As Integer
  Dim gefunden As Boolean, msg As String
  i = 0
  zahlen(0) = Int(45 * Rnd) + 1
  msg = zahlen (0)
  Do
    i = i + 1
    gefunden = False
    zahlen (i) = Int(45 * Rnd) + 1
    For j = 0 To i - 1
      If zahlen (j) = zahlen (i) Then
        i = i - 1
        gefunden = True
        Exit For
      End If
    Next j
    If Not gefunden Then msg = msg & ", " & zahlen (i)
  Loop Until i = 5
  MsgBox "CH - Lottozahlen = " & msg
End Sub
```

Speichern Sie die Mappe mit diesem Beispiel unter dem Namen: ***Meine_Mappe.XLS***

Exit For -Anweisung

Diese Anweisung bietet eine Möglichkeit zum Verlassen einer **For**-Schleife und kann nur in einer **For** ... **Next** oder **ForEach** ... **Next**-Schleife verwendet werden. **Exit For** hat zur Folge, dass die Ausführung mit der ersten Anweisung im Anschluss an die **Next**-Anweisung fortgesetzt wird. In verschachtelten **For**-Schleifen übergibt die **Next**-Anweisung die Steuerung an die Schleife der nächsthöheren (=aufrufenden) Verschachtelungsebene.

Exit definiert aber nicht das Ende einer Struktur. Verwechseln Sie die Exit-Anweisung nicht mit der End-Anweisung.
Im Zusammenhang mit einer **Do** ... **Loop** – Schleife kann die Anweisung **Exit Do** verwendet werden. Es gelten die entsprechenden Angaben.

Erläuterung zu Datenfeldern:

Datenfelder sind das, was man auch Arrays oder Vektoren nennt. Ein Datenfeld besteht aus mehreren Elementen des **gleichen** Typs. Die Anzahl der Elemente kann vom Programmierer frei gewählt werden.

Für Datenfelder ist die **Deklaration** über die **`Dim`** – Anweisung am Anfang der Prozedur oder Funktion oder auf Modulebene **zwingend**.

Bei der Deklaration werden der Name und die Dimension angegeben, wie im vorherigen Beispiel gezeigt wurde. Beim Datenfeld *zahlen* handelt es sich um ein Feld bestehend aus 6 Elementen: Die Numerierung der Plätze eines Datenfeldes fängt in Basic bei 0 an und nicht bei 1, wie in vielen anderen Sprachen. Die Felder können direkt über ihre Platznumerierung angesprochen werden, wie bei der Zuweisung im obigen Listing zu sehen ist.

Die Ordnungszahlen, die in einem Datenfeld an die Elemente angekoppelt sind, heissen Indizes. Die Indizes können auch frei gewählt werden:

```
Dim Woche(-3 To 3) As String   'Indizes sind hier -3, -2, -1, 0, 1, 2, 3
```

Die **Option Base – Anweisung** legt die erste gültige Zahl für die Indizes der Datenfelder eines Moduls fest. Wenn z.B: alle Datenfelder eines Moduls mit dem Index 1 statt 0 anfangen sollen, muss auf Modulebene die folgende Anweisung geschrieben werden:

```
Option Base 1
```

Felder können auch mehr als eine Dimension haben und damit zu mehrdimensionalen Matrizen werden. In dem Beispiel `matrix_test` hat die Matrix drei Zeilen und 2 Spalten. Ihren Elementen werden in den zwei verschachtelten For – Schleifen die Summen der Werte ihrer Indizes zugewiesen:

```
Sub matrix_test()
  Dim matrix(2, 1) As Integer, i As Integer, j As Integer
  For i = 0 To 2                'i Zeilenindex
    For j = 0 To 1              'j Spaltenindex
      matrix(i, j) = i + j
    Next j
  Next i
End Sub
```

For Each...Next - Anweisung

Die For Each - Anweisung erlaubt eine einfachere Schreibweise für Elemente von Datenfeldern bzw. Tabellen oder Arrays, wenn für jedes Element immer die gleiche Anweisung ausgeführt wird.

Syntax

```
For Each Element In Gruppe
  [Anweisungen]
  [Exit For]
  [Anweisungen]
Next [Element]
```

Element	Erforderlich – Variable zum Durchlauf durch die Elemente des Datenfeldes. Es ist nur eine Variable vom Typ Variant zulässig
Gruppe	Erforderlich – Name eines Datenfeldes
Anweisungen	Eine oder mehrere Anweisungen, die für jedes Element in Gruppe ausgeführt werden

Im Beispiel `TestForEach` werden unter dem Namen Feld drei Stringvariable definiert. Jede erhält einen bestimmten Teil eines Zitats:

```
Sub TestForEach()
  Dim Feld(2) As String
  Dim Zitat As Variant
  Feld(0) = "Kein Mensch kann das beim andern sehen"
  Feld(1) = "und verstehen was er nicht selbst"
  Feld(2) = "erlebt hat. (Hermann Hesse)"
  'Das Element, hier "Zitat", muss bei Datenfeldern vom
  'Typ Variant sein
  For Each Zitat In Feld
    MsgBox Zitat
  Next
End Sub
```

While — ***WHILE*-Schleife in *Pascal***

Diese Anweisung wird nur ganz kurz erwähnt, da die **Do ⋯ Loop** eine bessere Möglichkeit darstellt!

Die `While ⋯ Wend` - Anweisung führt eine Reihe von Anweisungen aus, solange eine gegebene Bedingung den Wert True hat.

Syntax

```
While Bedingung
  [Anweisungen]
Wend
```

Die Bedingung wird am Anfang der Schleife überprüft, d.h. wenn die Bedingung nicht wahr ist, kann die Schleife nicht ausgeführt werden. Die Ausführung der Schleife wird solange fortgesetzt, bis die Bedingung falsch ist.

```
Sub TestWhile()
  Dim x As Integer, y As Integer
  x = 5
  While x > 0
    y = y + x
    x = x - 1
  Wend
  MsgBox "x = " & x & "   y = " & y
End Sub
```

Funktionen

Zu den Unterprogrammen gehören auch Funktionen. Ein Funktionsunterprogramm ist eine Folge von Visual-Basic-Anweisungen, die durch die Anweisungen **`Function`** und **`End Function`** eingeschlossen sind. Zum Sprachumfang von Visual-Basic gehört eine grosse Anzahl von vordefinierten VBA-Funktionen. Es lassen sich auch benutzerdefinierte Funktionen erstellen. Damit kann man die Liste der in EXCEL verwendeten Tabellenfunktionen erweitern. Diese Funktionen erscheinen automatisch im Funktionsassistenten unter der Kategorie benutzerdefinierte Funktionen und können von dort aus bequem aufgerufen werden.

Vordefinierte Funktionen

Wenn man sich einen Überblick über die vorhandenen Funktionen verschaffen möchte, kann man sich die komplette Liste in der VBA-On-line- Hilfe ansehen:

Visual Basic – Editor → **?** → **INHALT/INDEX** → **INHALT**
→ **VISUALBASIC –SPRACHVERZEICHNIS** → **FUNTIONEN** → **A BIS Z**

Wir wollen in diesem Kapitel aus der grossen Anzahl der Funktionen die beiden Funktionen **MsgBox** und **InputBox**, die zur Standard- Ein / Ausgabe des Anwenders dienen, mit Hilfe von einigen Beispielen vorstellen.

Funktion MsgBox

Die Visual Basic Funktion **MsgBox** eignet sich besonders zur Anzeige von einfachen Meldungen. Sie kann auch für die Beantwortung von einfachen Fragen, also von Fragen mit Ja/Nein Antworten verwendet werden.

Die Funktion zeigt eine Meldung in einem Dialogfeld an und wartet darauf, dass der Benutzer auf eine Schaltfläche klickt. VBA koppelt an jede Schaltfläche einen numerischen Wert. Es wird ein Wert vom Typ Integer zurückgegeben, der anzeigt, auf welche Schaltfläche der Benutzer geklickt hat. z.B. liefert die Schaltfläche **JA** – wenn sie angeklickt wird – den Wert 6, die Schaltfläche **NEIN** den Wert 7.

Wird der Rückgabewert der Funktion verwendet, müssen die Argumente in Klammern gesetzt werden – wird der Rückgabewert der Funktion nicht verwendet, dürfen keine Klammern gesetzt werden.

Syntax

```
MsgBox ( Prompt, Buttons, Titel)
```

Prompt: mit diesem Argument wird die Meldung angegeben, die angezeigt werden soll

Buttons: Der Standardwert ist **`vbOKOnly`**, d.h. nur Schaltfläche OK anzeigen
Es können verschiedene vordefinierte Konstanten angegeben werden z.B.:
- **`vbYesNo`** für die 2 Schaltflächen Ja (6) Nein (7) *(Rückgabewerte in ( ))*
- **`vbYesNoCancel`** für die 3 Schaltflächen Ja (6) Nein (7) Abbrechen (2)

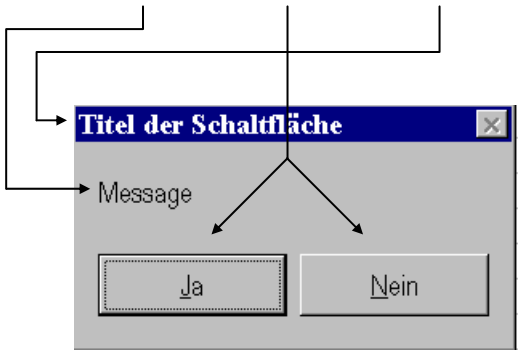
Titel: bewirkt Titel in der Titelleiste der Box

Eine der einfachsten Anweisungen in VBA ist die Ausgabe einer eigenen Meldung am Bildschirm:

```
MsgBox "FHBB - Fachhochschule beider Basel"
```

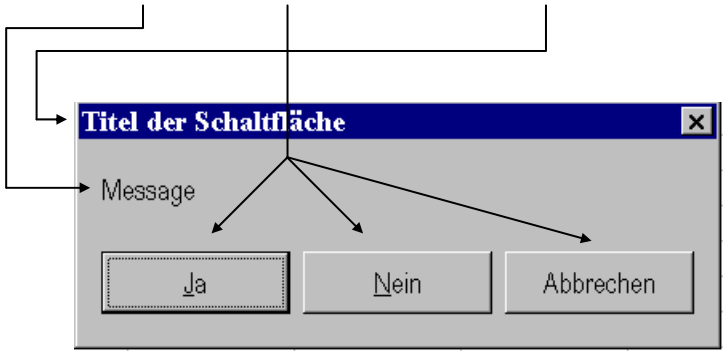
Im Kapitel 2 wurde im Beispiel 3 die Verwendung der Funktion **MsgBox** mit dem Parameter **vbYesNo** bereits gezeigt:

```
replay = MsgBox(msg, vbYesNo, titel)
```



Am Ende dieses Kapitels zeigen wir im Beispiel 8 die Verwendung der Funktion **MsgBox** mit dem Parameter **vbYesNoCancel**:

```
replay = MsgBox(msg, vbYesNoCancel, titel)
```



Für das Argument Buttons gibt es neben den erwähnten Einstellungen **vbYesNoCancel** und **vbYesNo** noch weitere Einstellungen:

vbOKOnly	Nur die Schaltfläche OK anzeigen.
vbOKCancel	Schaltflächen OK und Abbrechen anzeigen.
vbAbortRetryIgnore	Schaltflächen Abbruch, Wiederholen und Ignorieren anzeigen.
vbYesNoCancel	Schaltflächen Ja, Nein und Abbrechen anzeigen.
vbYesNo	Schaltflächen Ja und Nein anzeigen.
vbRetryCancel	Schaltflächen Wiederholen und Abbrechen anzeigen.

vbCritical	Meldung mit Stop-Symbol anzeigen.
vbQuestion	Meldung mit Fragezeichen-Symbol anzeigen.
vbExclamation	Meldung mit Ausrufezeichen-Symbol anzeigen.
vbInformation	Meldung mit Info-Symbol anzeigen.

Rückgabewerte:

vbOK	1	OK
vbCancel	2	Abbrechen
vbAbort	3	Abbruch
vbRetry	4	Wiederholen
vbIgnore	5	Ignorieren
vbYes	6	Ja
vbNo	7	Nein

Beispiele zur Funktion MsgBox

1. Beispiel

Dieses Beispiel zeigt die Möglichkeit den ASCII-Code eines Zeichens anzuzeigen.

```
Sub zeichen_code_ausgeben()
  Dim msg As String
  msg = "Das Zeichen A hat den ASCII-Code " & Asc("A")
  MsgBox msg
End Sub
```

2. Beispiel

Dieses Beispiel zeigt, wie man auf dem Bildschirm einen Zeilenumbruch erhalten kann.

```
Sub neue_zeile()
  Dim msg As String
  msg = "Hallo User, " & Chr(13) & Chr(10) & _
  "Wie geht's heute?"
  MsgBox msg
End Sub
```

3. Beispiel

Im folgenden Beispiel wird die ausführbare Anweisung **MsgBox** über drei Zeilen fortgesetzt, die Variablendeklaration und die Wertzuweisung stehen in einer Zeile und ausserdem wird die Verwendung von Kommentarzeilen gezeigt. Beachten Sie den Titel der Meldungsbox und ganz speziell auch die Wirkung der Konstanten **vbExclamation**.

```
Sub DemoFeld()     ' Diese Prozedur deklariert eine
  ' Zeichenfolgenvariable, weist ihr den
  ' Wert Lehni zu und zeigt dann
  ' die zusammengesetzte Meldung an.
  Dim meineVar As String : meineVar = "Lehni"
  MsgBox Prompt:="Hallo " & meineVar, _
                 Title:="Begrüssungsfeld", _
                 Buttons:=vbExclamation
End Sub
```

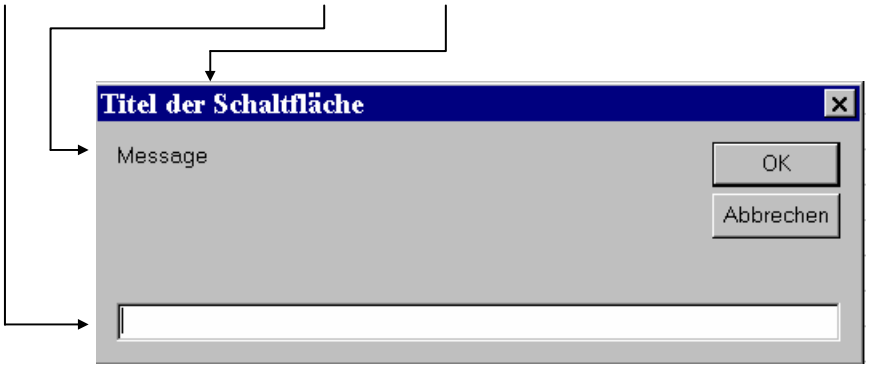
Funktion InputBox

Die Funktion **`InputBox`** dient zur Eingabe von Zahlen oder Text. Die Funktion zeigt eine Eingabeaufforderung in einem Dialogfeld an, wartet auf die Eingabe eines Textes oder auf das Klicken auf eine Schaltfläche und gibt einen Wert vom Typ String zurück, der den Inhalt des Textfeldes angibt.

Syntax InputBox **(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])**

prompt	*Erforderlich. Ein Zeichenfolgenausdruck, der als Meldung im Dialogfeld erscheint. Die Maximallänge von prompt ist - je nach Breite der verwendeten Zeichen - etwa 1024 Zeichen. Wenn prompt aus mehreren Zeilen besteht, müssen Sie die Zeilen mit einem Wagenrücklaufzeichen (Chr(13)), einem Zeilenvorschubzeichen (Chr(10)) oder einer Kombination aus Wagenrücklaufzeichen und Zeilenvorschubzeichen (Chr(13) & Chr(10)) trennen.*
title	*Optional. Ein Zeichenfolgenausdruck, der in der Titelleiste des Dialogfeldes angezeigt wird. Wenn Sie title nicht angeben, wird der Anwendungsname in der Titelleiste angezeigt.*
default	*Optional. Ein Zeichenfolgenausdruck, der als Voreinstellung im Textfeld angezeigt wird, wenn der Benutzer keine Eingabe vorgenommen hat. Wenn Sie default nicht angeben, wird das Textfeld ohne Text angezeigt.*
xpos	*Optional. Ein numerischer Ausdruck, der den horizontalen Abstand (in Twips) des linken Rands des Dialogfeldes vom linken Rand des Bildschirms festlegt. Wenn Sie xpos nicht angeben, wird das Dialogfeld horizontal zentriert.*
ypos	*Optional. Ein numerischer Ausdruck, der den vertikalen Abstand (in Twips) des oberen Rands des Dialogfeldes vom oberen Rand des Bildschirms festlegt. Wenn Sie ypos nicht angeben, wird das Dialogfeld etwa ein Drittel unterhalb des oberen Bildschirmrands (bezogen auf die gesamte Bildschirmhöhe) angezeigt.*
helpfile	*Optional. Ein Zeichenfolgenausdruck, der die Hilfedatei mit der kontextbezogenen Hilfe für das Dialogfeld angibt. Wenn Sie helpfile angeben, müssen Sie auch context angeben.*
context	*Optional. Ein numerischer Ausdruck mit der Hilfekontextkennung, die der Autor der Hilfe für das entsprechende Hilfethema vergeben hat. Wenn Sie context angeben, müssen Sie auch helpfile angeben.*

```
eingabe = InputBox(msg, Titel)
```



Beispiele zur Funktion InputBox

4. Beispiel

Beispiel zur Eingabe eines Textes:

```
Sub frage()
  Dim IhrName As String
  IhrName = InputBox("Wie heissen Sie?")
  MsgBox "Ihr Name lautet: " & Chr(13) & Chr(10) & IhrName
End Sub
```

5. Beispiel Zwei Beispiele zur Eingabe einer Zahl:

Eingabe ohne Definition des Datentyps – es erfolgt keine Fehlermeldung

```
Sub zahleinlesen()
  Dim titel As String
  titel = "Eingabe ohne Kontrolle - keine Fehlermeldung"
  zahl = InputBox("Eingabe einer Zahl bitte: ", titel)
  MsgBox zahl
End Sub
```

Eingabe einer Zahl – Datentyp Byte- beachten Sie dabei die möglichen Fehlermeldungen

```
Sub zahleinlesen1()
  Dim titel As String, zahl As Byte
  titel = "Eingabe ohne Kontrolle aber Fehlermeldung"
  zahl = InputBox("Eingabe einer Zahl bitte: ", titel)
  MsgBox zahl
End Sub
```

- *falls Sie einen **Text** eingeben, erscheint die Fehlermeldung **Typen unverträglich***
- *falls Sie einen negativen Wert oder eine Zahl > 255 eingeben, erscheint die Fehlermeldung **Überlauf***
- *aber Achtung: bei Zahl mit Dezimalpunkt (0 bis 254**) keine** Fehlermeldung in Basic!!!*

Um richtige Eingabewerte zu gewährleisten, muss der Rückgabewert der Funktion überprüft werden. Zur Kontrolle, ob eine gültige Eingabe vorliegt, kann man die beiden Funktionen **IsNumeric** und **IsDate** verwenden**,** die zur Überprüfung von Zeichenketten zur Verfügung stehen:

- **IsDate** liefert "Wahr", wenn die Zeichenkette ein gültiges Datumsformat besitzt, sonst "Falsch"
- **IsNumeric** liefert "Wahr", wenn die Zeichenkette ein gültiges Zahlenformat besitzt.
  Wenn die Eingabe kein numerisches Format hat, gibt es drei Möglichkeiten:
  - der Anwender hat mindestens einen Buchstaben oder ein Sonderzeichen getippt, die Eingabe ist dann nicht leer, Anzeige einer Fehlermeldung
  - der Anwender hat nichts eingegeben und die Schaltfläche OK geklickt
  - der Anwender hat die Schaltfläche Abbrechen betätigt, unabhängig von der vorherigen Eingabe.

6.Beispiel

Beispiel zur Eingabe eines **Datums mit Kontrolle**. Es soll ein Datum im angegebenen Format eingegeben werden

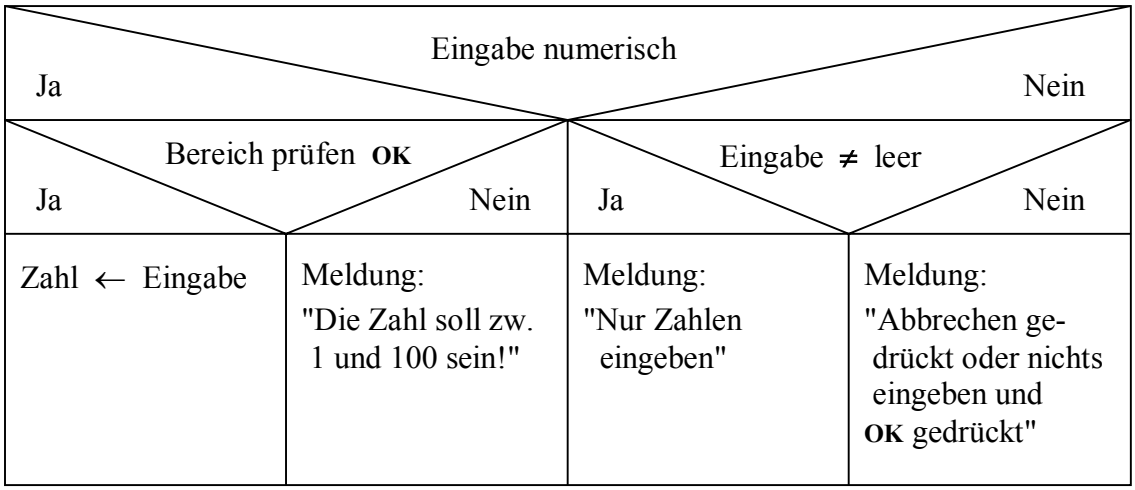
```
Sub Eingabetest()
  Dim neudatum As String
  neudatum = InputBox("Geben Sie einen Monat im Format mmm-jj ein:")
  If neudatum <> "" Then
    If IsDate(neudatum) Then
      MsgBox "Eingabe in Ordnung"
    Else
      MsgBox "Eingabe war kein Datum"
    End If
  Else
    MsgBox "Abbrechen gedrückt"
  End If
End Sub
```

7.Beispiel

Beispiel zur Eingabe einer Zahl zwischen 1 und 100 mit Kontrolle. Wenn Buchstaben oder andere Zeichen eingegeben werden, soll eine Fehlermeldung erscheinen. Durch Klicken auf die Schaltfläche *Abbrechen*, kann die Eingabe zu jedem Zeitpunkt beendet werden.

Das Eingabefenster soll so lange am Bildschirm erscheinen, bis eine Zahl zwischen 1 und 100 eingegeben wird. Das wird durch Verwendung einer Schleife erreicht. Beachten Sie auch das beigefügte Struktogramm, das Ihnen den Überblick über die Eingabekontrolle zeigt.

```
Sub LeseZahlEin()
  Dim eingabe As String, msg As String,
  Dim titel As String, zahl As Single
  titel = "Eine Zahl Eingeben"
  Do
    msg = "Bitte eine Zahl zwischen 1-100 eingeben: "
    eingabe = InputBox(msg, titel)
    If IsNumeric(eingabe) Then
      If eingabe <= 100 And eingabe > 0 Then
        zahl = eingabe
      Else
        MsgBox "Die Zahl soll zw. 1 und 100 sein!"
      End If
    ElseIf eingabe <> "" Then
          MsgBox "Nur Zahlen eingeben!"
        Else
          MsgBox "Abbrechen gedrückt oder nichts eingegeben und ok gedrückt"
    End If
  Loop Until IsNumeric(eingabe) And zahl > 0 And zahl <= 100
  msg = "der eingegebene Wert war " & zahl
  MsgBox msg
  zahl_gerundet = CInt(zahl)
  If zahl <> zahl_gerundet Then
    msg = "der gerundete Wert ist " & zahl_gerundet
    MsgBox msg
  End If
End Sub
```



Beachten Sie, dass man zum Runden der Zahl die Funktion **CInt** (Convert Integer – Umwandlung in ganze Zahl) verwenden muss.

Bemerkung: Die Funktionen Int und Fix wandeln auch reelle Zahlen in ganze Zahlen um. Die Beispiele zeigen, wie dabei gerundet wird:

Int(9.2), Int(9.8), Fix(9.2) und Fix(9.8) liefern das Ergebnis 9

Int(-9.2), Int(-9.8) liefern das Ergebnis –10

Fix(-9.2) und Fix(-9.8) liefern das Ergebnis -9

Von den weitere Umwandlungsfunktionen erwähnen wir noch:

CCur	wandelt in Währung um
Cdate	wandelt in Datum/Zeit um
Hex	wandelt ein Zahl in Hexadezimal um

8.Beispiel

Dieses Beispiel zeigt die Möglichkeit, für ein 3D-Diagramm auch nach der Erstellung mit dem Diagrammassistenten Beschriftungen einzugeben.

```
Sub BeschriftungenEingeben()
  Dim ButtonNr As Byte
  Title = "Beschriftung des Diagramms"
  msg = "Soll das Diagramm einen Titel und Achse-Beschriftungen haben?"
  ButtonNr = MsgBox(msg, vbYesNoCancel, Title)
  Select Case ButtonNr
    Case 2: MsgBox "Abbrechen gewählt"
                              'Schaltfläche Abbrechen, Programmende
    Case 6: msg = "Bitte einen Titel eingeben:"     'Schaltfläche Ja
            Antwort = InputBox(msg, Title)
            If Antwort <> "" Then DiagTitel = Antwort
            msg = "Bitte den Text für die X-Achse eingeben:"
            Antwort = InputBox(msg, Title)
            If Antwort <> "" Then X = Antwort
            msg = "Bitte den Text für die Y-Achse eingeben:"
            Antwort = InputBox(msg, Title)
            If Antwort <> "" Then Y = Antwort
            msg = "Bitte den Text für die Z-Achse eingeben:"
            Antwort = InputBox(msg, Title)
            If Antwort <> "" Then Z = Antwort
    Case 7: MsgBox "Nein gewählt"                'Schaltfläche Nein
  End Select
End Sub
```

Wegen der VBA-Konstanten **vbYesNoCancel** erscheinen in dem von der **MsgBox**-Funktion erzeugten Dialogfenster drei Schaltflächen. Diese sind, wie der Name der Konstanten verrät **Ja**, **Nein** und **Abbrechen**. Wegen der drei Schaltflächen ist die Anzahl der möglichen Rückgabewerte der Funktion **MsgBox** drei. Man erhält die Zahlen: **2** für Abbrechen, **6** für Ja und **7** für Nein.
Mit Hilfe einer Select-Anweisung wird der Rückgabewert ausgewertet. Im Fall Nein und Abbrechen wird nur eine Meldung angezeigt, nur der Ja-Fall ist ausprogrammiert. Bei den verwendeten Eingaben mit der Funktion **InputBox** kann auch das Eingabefeld leergelassen werden oder man kann die Eingabe abbrechen.

Benutzerdefinierte Funktionen

Eine Function-Prozedur ist eine Folge von VBA-Anweisungen, die in den Anweisungen **Function** und **End Function** eingeschlossen sind. Der Name der Funktion unterliegt den gleichen Regeln wie die Variablendeklaration. Die Funktion kann Argumente, wie z.B. Konstanten, Variablen oder Ausdrücke verwenden, die über die aufgerufene Prozedur übergeben werden. Wenn sie über keine Argumente verfügt, muss deren Function-Anweisung ein leeres Klammernpaar enthalten.

Benutzerdefinierte Funktionen haben folgender Aufbau:

```
Function name (variable1; variable2; ...)
  Berechnung(en)  bzw.  Anweisung(en)
  name = .....
End Funktion
```

Eine Funktion gibt einen Wert zurück, indem ihrem Namen **name** ein Wert in einer oder mehreren Anweisungen der Prozedur zugewiesen wird.

Die detaillierte Beschreibung der Syntax finden Sie in der VBE – Hilfe → Menüpunkt:
→ **?** → **VB-HILFE** → **INDEX** → **Function** eintippen → **Suchen**
→ **Function-Anweisung** anklicken

Beispiel: Erstellung der Funktionen zum Umrechnen der Temperaturen von Celsius in Fahrenheit und umgekehrt.

1. *Funktion erstellen*

   Öffnen Sie Ihre Datei mit dem Namen ***Meine_Mappe.XLS*** (siehe Kapitel 4)
   Öffnen Sie den VisualBasic-Editor:
   → **EXTRAS** → **MAKRO** → **VISUAL BASIC-EDITOR** *oder* Symbol anklicken

2. Schreiben Sie in das Modulblatt die folgende Subprozedur ***Main_Temperatur*** und die beiden Funktions-Prozeduren ***Celsius*** und ***Fahrenheit***.

```
Sub Main_Temperatur()
  temp = InputBox("Geben Sie die Temperatur in Fahrenheit ein.")
  If temp = "" Then
    MsgBox "Abbrechen gedrückt oder nichts eingegeben und ok gedrückt"
  Else
    MsgBox " Temperatur : " & Format(Celsius(temp), "0.0") & " °Celsius."
  End If
End Sub

Function Celsius(GradFahrenheit)
  Celsius = (GradFahrenheit - 32) * 5 / 9
End Function

Function Fahrenheit(GradCelsius)
  Fahrenheit = 32 + 9 / 5 * GradCelsius
End Function
```

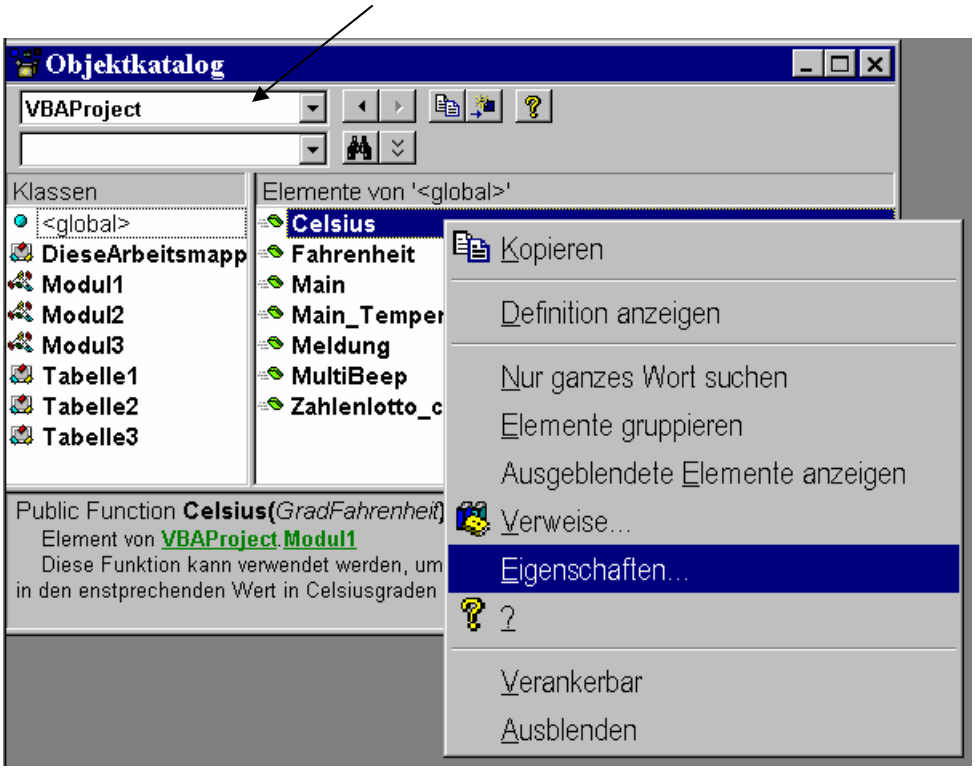
Im obenstehenden Beispiel berechnet die Funktion mit Namen **Celsius**: Celsius-Grade aus Fahrenheit-Graden und die zweite Funktion rechnet umgekehrt. Wenn die Funktion aus der Main-Prozedur aufgerufen wird, wird eine den Argumentwert enthaltende Variable der Funktion übergeben. Das Ergebnis der Berechnung wird an die aufrufende Prozedur zurückgegeben und in einem Meldungsfeld angezeigt.

3. Aktivieren Sie die Excel Tabelle und testen Sie die erstellten Funktionen:

   a) Durch Aufruf der Prozedur ***Main_Temperatur***
   → **EXTRAS** → **MAKRO** → **MAKROS** → ***Main_Temperatur*** → **Ausführen**

   b) Geben Sie in der Spalte A die Celsius Temperaturwerte -10 bis 40 mit der Schrittweite 2 ein und berechnen Sie in der Spalte B die entsprechenden Fahrenheit Temperaturwerte, in dem Sie die Funktion ***Fahrenheit*** mit dem Funktions-Assistenten aus der Kategorie *Benutzerdefiniert* auswählen. Erzeugen Sie anschliessend in der Spalte C Fahrenheit Temperaturen von 10 bis 100 mit Schrittwerte 5 und in Spalte D die entsprechenden Celsius Temperaturwerte.

4. Wenn die erstellte Funktion getestet wurde und in Ordnung ist, empfiehlt es sich, diese Funktion für den Aufruf über den Funktions-Assistenten mit einer Beschreibung zu versehen. Aktivieren Sie VBE und → **ANSICHT** → **OBJEKTKATALOG**. Wählen Sie den Listeneintrag **VBAProject** anstelle von <Alle Bibliotheken> und markieren sie die gewünschte Funktion z.B. **Celsius**. Öffnen Sie das Kontextmenü → **EIGENSCHAFTEN**. Sie erhalten das Dialogfenster Elementoptionen und können im Feld Beschreibung einen ausführlichen Kommentar zur Funktion eingeben.



5. Geben Sie der Tabelle den Namen *Temperatur* und speichern Sie die Mappe.

Bemerkungen

- Die Verwendung von benutzerdefinierten Funktionen ist von EXCEL aus sehr bequem, da diese Funktionen automatisch im Funktionsassistenten unter benutzerdefinierten Funktionen erscheinen und von dort aus aufgerufen werden können. Dieses gilt nur wenn die entsprechende Mappe geöffnet ist.

- Der Aufruf einer Function-Prozedur innerhalb eines Ausdrucks erfolgt durch Angabe des Funktionsnamens gefolgt von der Argumentliste in Klammern. Im Gegensatz zu einer Sub-Prozedur kann eine Function-Prozedur auf der rechten Seite eines Ausdrucks verwendet werden. Dadurch besteht die Möglichkeit, den Rückgabewert der Function-Prozedur in gleicher Weise wie den Rückgabewert einer Standard-Funktion, wie zum Beispiel `Cos`, weiterzuverarbeiten.

- Variablen in Function-Prozeduren können entweder explizit deklariert sein oder ohne Deklaration verwendet werden. Explizit über Dim-Anweisungen deklarierte Variablen sind innerhalb dieser Funktions-Prozedur immer lokal. Variablen, die in einer Prozedur verwendet, aber nicht dort deklariert werden, sind auch lokal, sofern sie nicht bereits explizit auf einer höheren Ebene ausserhalb der Prozedur deklariert wurden. Um Namenskonflikte zu vermeiden, verwenden Sie die explizite Variablendeklaration. Mit der `Option Explicit`-Anweisung können Sie die explizite Deklaration von Variablen erzwingen.

- Innerhalb einer Function-Prozedur oder einer Sub-Prozedur kann keine weitere Function-Prozedur definiert werden. Function-Prozeduren sind standardmässig öffentlich, wenn sie nicht explizit mit **`Private`** festgelegt werden. Wird **`Static`** nicht angegeben, so bleiben die Werte lokaler Variablen zwischen den Aufrufen einer Funktion nicht erhalten.

- Die `Exit Function`-Anweisung führt zum unmittelbaren Verlassen einer Function-Prozedur. Die Programmausführung wird mit der Anweisung fortgesetzt, die auf die Anweisung folgt, welche die Function-Prozedur aufgerufen hat. `Exit Function`-Anweisungen können an beliebigen Stellen innerhalb einer Function-Prozedur verwendet werden.

Unterprogramme

VBA bietet wie auch andere höhere Programmiersprachen die Möglichkeit, Unterprogramme zu verwenden. Diese Art des Programmierens erlaubt eine Aufteilung einer komplexen Aufgabe in einzelne Teilaufgaben, die durch selbständige Programmteile gelöst werden können. Diese Programmteile können extra getestet werden, ausserdem können solche Programmteile in verschiedenen Programmen verwendet werden. Eine Prozedur zum Sortieren von Namen könnte z.B. in zwei verschiedene Programme eingebunden und von beiden benutzt werden. Wir werden in diesem Kapitel eine komplexe Aufgabe kennenlernen, die nur aus Aufrufen von Prozeduren besteht.

Es gibt in VBA zwei verschiedene Arten von Unterprogrammen:

Funktionen

Prozeduren

Prozeduren und Funktionen bieten die Möglichkeit, Anweisungsfolgen mit einem Namen zu versehen und mit diesem Namen aufzurufen. Beim Aufruf können von Fall zu Fall verschiedene Parameter übergeben werden.

Sub-Prozedur

Eine Sub-Prozedur ist eine Folge von VBA-Anweisungen, die in den Anweisungen **`Sub`** und **`End Sub`** eingeschlossen sind und Aktionen ausführen, aber keinen Wert zurückgeben. Eine Sub-Prozedur kann Argumente, z.B. Konstanten, Variablen oder Ausdrücke verwenden, die über eine aufrufende Prozedur übergeben werden. Wenn eine Sub-Prozedur über keine Argumente verfügt, muss die Sub-Anweisung ein leeres Klammernpaar enthalten.

Beispiel aus der Hilfe:

Das Beispiel wird ausführlich durch Kommentarzeilen erklärt

```
          ' Prozedur namens HoleInfo deklarieren
          ' Diese Sub-Prozedur besitzt keine Argumente
Sub HoleInfo()
          ' Zeichenfolge namens Antwort deklarieren
  Dim Antwort As String
          ' Antwort dem Rückgabewert der InputBox-Funktion zuweisen
  Antwort = InputBox(Prompt:="Wie heissen Sie?")
          ' Bedingte If...Then...Else-Anweisung
  If Antwort = Empty Then
          ' Ruft die MsgBox-Funktion auf
    MsgBox Prompt:="Sie haben keinen Namen eingegeben."
  Else
          ' MsgBox-Funktion gibt Text aus, der mit dem
          ' Inhalt der Variablen "Antwort" verkettet ist
    MsgBox Prompt:="Ihr Name lautet " & Antwort
          ' If...Then...Else-Anweisung abschliessen
  End If
          ' Sub-Prozedur abschliessen
End Sub
```

Syntax:

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

Public /Private	Optional; wenn keine dieser Optionen angegeben wird, gilt implizit Public, d.h. eine Prozedur kann von Unterprogrammen des gleichen Moduls oder anderer Module (also aus allen Modulen eines Projekts) aufgerufen und benutzt werden. Das Schlüsselwort Private schränkt den Wirkungsbereich des Unterprogramms auf das Modul ein, in dem es deklariert worden ist. Wenn eine Prozedur als Public gilt, kann sie sogar aus Modulen anderer Projekte aufgerufen werden.
Static	Optional; die lokalen Variablen der Sub-Prozedur bleiben zwischen Aufrufen erhalten. Das Attribut Static wirkt sich nicht auf Variablen aus, die ausserhalb der Sub-Prozedur deklariert wurden, auch wenn sie in der Prozedur verwendet werden. Wird Static nicht angegeben, so werden die Werte von lokalen Variablen wieder auf 0 gesetzt bei Zahlen und auf "" bei Zeichenketten.
Name	Erforderlich; Name der Sub-Prozedur gemäss den Standardkonventionen für Namen von Variablen.
ArgListe	Optional; Variablenliste mit den Argumenten, die an die Sub-Prozedur beim Aufruf übergeben werden. Mehrere Variablen werden durch Kommas getrennt.
Anweisungen	Optional; eine beliebige Gruppe auszuführender Anweisungen im Rumpf der Sub-Prozedur.

Syntax ArgListe:

[Optional] [ByVal | ByRef] [ParamArray] VarName[( )] [As Typ] [= Standardwert]

Optional	Optional; Schlüsselwort, das angibt, dass ein Argument nicht erforderlich ist. Alle im Anschluss an Optional in ArgListe angegebenen Argumente müssen auch optional sein und mit dem Schlüsselwort Optional deklariert werden. Optional kann nicht verwendet werden, wenn ParamArray verwendet wird
ByVal	**Optional; Das Argument wird als Wert übergeben**
ByRef	**Optional; Das Argument wird als Referenz übergeben. ByRef ist der Standard in Visual Basic**
ParamArray	Optional; Ist nur als letztes Argument in ArgListe zulässig und gibt an, dass das letzte Element ein als Optional deklariertes Datenfeld mit Variant-Elementen ist. Das Schlüsselwort ParamArray erlaubt die Angabe einer variablen Anzahl von Argumenten und darf nicht in Kombination mit den Schlüsselwörtern ByVal, ByRef oder Optional verwendet werden
VarName	Erforderlich; Name der Variablen, die das Argument darstellt, gemäss den Standardkonventionen für Namen von Variablen
Typ	Optional; Datentyp des an die Prozedur zu übergebenden Arguments. Zulässige Typen sind: Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (zur Zeit nicht unterstützt), Date, String (nur Zeichenfolgen variabler Länge), Object, Variant. Ist der Parameter nicht Optional, kann ein benutzerdefinierter Typ oder ein Objekttyp angegeben werden

Standardwert	Optional; eine beliebige Konstante oder ein Konstantenausdruck. Nur gültig, wenn der Parameter Optional entspricht. Ist der Typ Object, so kann der explizite Standardwert nur Nothing sein

Bemerkungen

- Sub-Prozeduren sind Public (öffentlich), wenn sie nicht mit Private festgelegt werden. Wird `Static` nicht angegeben, so bleiben die Werte lokaler Variablen zwischen den Aufrufen nicht erhalten.

- Der gesamte ausführbare Code muss sich in Prozeduren befinden. Innerhalb einer `Sub` – oder `Function` – Prozedur kann keine weitere `Sub`-Prozedur definiert werden.

- Das Schlüsselwort **`Exit Sub`** führt zum unmittelbaren Verlassen einer Sub-Prozedur. Die Programmausführung wird mit der Anweisung fortgesetzt, die auf die Anweisung folgt, die die Sub-Prozedur aufgerufen hat. Exit Sub-Anweisungen können beliebig oft an beliebigen Stellen innerhalb einer Sub-Prozedur verwendet werden.

- Wie bei einer Function-Prozedur handelt es sich bei der Sub-Prozedur um eine eigenständige Prozedur, die Argumente erhalten, eine Reihe von Anweisungen ausführen und die Werte der übergebenen Argumente ändern kann. Im Gegensatz zu einer Function-Prozedur, die einen Wert zurückgibt, kann eine Sub-Prozedur aber nicht in einem Ausdruck verwendet werden. Der Aufruf einer Sub-Prozedur erfolgt durch Angabe des Prozedurnamens gefolgt von der Argumentliste.

- Variablen in Sub-Prozeduren können entweder explizit deklariert oder ohne Deklaration verwendet werden. Explizit (über Dim oder eine entsprechende Anweisung) in einer Prozedur deklarierte Variablen sind innerhalb dieser Prozedur immer lokal. Variablen, die in einer Prozedur verwendet, aber nicht dort deklariert werden, sind auch lokal, sofern sie nicht bereits explizit auf einer höheren Ebene ausserhalb der Prozedur deklariert wurden.

- Vorsicht: Eine Prozedur kann eine nicht explizit in der Prozedur deklarierte Variable zwar verwenden, aber es kann zu Namenskonflikten kommen, falls ein anderes auf Modulebene definiertes Element denselben Namen hat. Verwendet Ihre Prozedur eine nicht deklarierte Variable mit einem Namen, der dem Namen einer anderen Prozedur, Konstanten oder Variablen entspricht, so wird die Variable als Verweis auf den Namen interpretiert, der auf Modulebene definiert wurde. **Deklarieren Sie Variablen explizit, um derartige Konflikte zu vermeiden. Mit der Anweisung `Option Explicit` können Sie die explizite Deklaration von Variablen erzwingen.**

- Anmerkung: Eine Sub-Prozedur können Sie nicht mit GoSub, GoTo oder Return aufrufen oder verlassen.

Mit Makros komplexe Aufgaben ausführen

Bei der Erstellung eines Makros zur Ausführung einer komplexen Aufgabe geht man so vor, dass man die Aufgabe aufteilt, für jeden Teil ein Makro erstellt und dann die Teile wieder zu einer Einheit zusammenfügt. In den Programmiersprachen spricht man von modularer Programmierung (Unterprogrammtechnik).

Beispiel aus der Auftragsverwaltung:

Es stehen uns Daten der Firma Müller Textilien in der Excel-Datei *Aufträge.XLS* zur Verfügung. Diese Datei enthält 3582 Datensätze mit Angaben ab Nov 94 bis Dez 97. Sie sehen im folgenden einen kleinen Ausschnitt dieser grossen Tabelle.

DATUM	STAAT	KANAL	PREIS	KATEGORIE	MENGE	NETTO	LISTE	EVK-PREIS
Nov 94	WA	Großhandel	Hoch	München	40	110.00		
Nov 94	WA	Großhandel	Hoch	Kunst	25	68.75		
Nov 94	WA	Einzelhandel	Hoch	Kunst	3	16.50		
Nov 94	WA	Einzelhandel	Niedrig	Umwelt	50	175.00		
Nov 94	WA	Großhandel	Niedrig	Dinosaurier	40	70.00		

Diese Daten sind jeden Monat um die neuen Angaben zu ergänzen. Diese Datensätze sind in der Textdatei *monat.txt* abgespeichert. Der Beginn dieser Textdatei für den Monat Januar 1998 sieht wie folgt aus:

Müllers Textilien
Aufträge vom Januar 1998

Staat	Kanal	Preis	Kategorie	Menge	Netto	Liste	EVK-Preis
CH	Einzelhandel	Mittel	Kinder	9	40.5	4.5	40.5
		Niedrig		143	434.06	3.5	500.5
		Hoch	Kunst	17	93.5	5.5	93.5
		Mittel		23	103.5	4.5	103.

Wir wollen ein VBA-Programm erstellen, dass die Daten des neuen Monats, die in der Textdatei monat.txt gespeichert sind in eine EXCEL-Tabelle importiert wird und an die bestehende Tabelle *aufträge.XLS* angehängt wird. Bevor wir die Daten anfügen können, müssen einige Anpassungen vorgenommen werden.

Bei der Lösung der Aufgabe werden wir das Problem in fünf Teilschritte zerlegen, für jeden Teilschritt ein Makro erstellen und testen und zuletzt die fünf Makros in einem einzigen Programm zusammenfassen. Wir unterscheiden folgende Teile:

1. Teil:	Textdatei mit Angaben für den neuen Monat in eine EXCEL–Tabelle importieren	*DateiImportieren*
2. Teil:	Beschriftungen ergänzen	*BeschriftungEinfügen*
3. Teil:	Aktuelles Datum einfügen	*DatumEinfügen*
4. Teil:	Die neuen Monatsdaten an die bestehenden Auftragsdaten anfügen	*DatenbankAnhängen*
5. Teil:	Importierte Textdatei löschen	*TabelleLöschen*

Da die Lösung dieser Aufgabe einige neue Möglichkeiten von EXCEL zeigt, empfehlen wir Ihnen, die 5 Teile zuerst **ohne Makroaufzeichnung** auszuprobieren.

Wenn Sie die Aufzeichnung durchführen, machen Sie alle 5 Schritte in **einer** EXCEL-Sitzung.

*Kopieren Sie die Dateien **Monat.txt** und **Aufträge.xls** in Ihr Verzeichnis …*

1. Teil: Textdatei importieren

Die Aufträge des aktuellen Monats – in unserem Beispiel Januar 1998 – sind in der Textdatei ***Monat.txt*** abgelegt. Es ist diese Datei zu öffnen – der Inhalt ist bereits in Spalten aufgeteilt – und in die Excel Arbeitsmappe zu verschieben.

1. Öffnen Sie eine neue Mappe und speichern Sie diese unter dem Namen ***Hilfe.XLS***.
2. → **MAKRO AUFZEICHNEN**, Makroname: *DateiImportieren*
3. → **DATEI** → **ÖFFNEN** geben Sie als Dateiname *Monat.txt* ein und klicken Sie auf Öffnen. Der Text-Assistent wird eingeblendet. Ändern Sie den Wert von *Import beginnen* in Zeile **4** → **WEITER.** Entfernen Sie in der Vorschau der markierten Daten den Strich mit Pfeil bei Position 47 durch Doppelklicken → **ENDE.** Die Textdatei wird geöffnet und die Spalten sind in Excel-Spalten aufgeteilt.
4. Verschieben Sie die Tabelle *Monat* vor die Tabelle1 der aktuellen Mappe *Hilfe.xls* → **BEARBEITEN** → **BLATT VERSCHIEBEN / KOPIEREN** . Die Mappe monat.txt wird ausgeblendet.
5. Löschen Sie die Zeile 2 mit den Gleichheitszeichen und aktivieren Sie die Zelle A1
6. → **AUFZEICHNUNG BEENDEN**
7. Wechseln Sie zu Visual Basic und sehen Sie sich das aufgezeichnete Makro an.
8. Fügen Sie vor der Anweisung `Workbooks.OpenText` eine folgende neue Zeile ein: `neuDatei = Application.GetOpenFilename("Textdateien,*.txt")` Das Wort *neuDatei* ist eine Variable, in der der angegebene Dateiname gespeichert wird. Ersetzen Sie in der Anweisung `Workbooks.OpenText` den gesamten Dateinamen einschliesslich der Anführungszeichen durch *neuDatei*.
9. Ändern Sie die Wörter `Sheets("Monat").Move` in **`ActiveSheet.Move`**
10. Vergleichen Sie Ihr Makro mit dem unten abgebildeten. Falls Unterschiede bestehen, führen Sie die nötigen Änderungen durch und testen Sie zum Schluss Ihr Makro.

```
Sub DateiImportieren()
  neuDatei = Application.GetOpenFilename("Textdateien,*.txt")
  Workbooks.OpenText FileName:=neuDatei, _
    Origin:=xlWindows, StartRow:=4, _
    DataType:=xlFixedWidth, _
    FieldInfo:=Array(Array(0, 1), Array(5, 1), _
    Array(20, 1), Array(28, 1), Array(44, 1), _
    Array(52, 1), Array(61, 1), Array(68, 1))
  ActiveSheet.Move Before:=Workbooks("Hilfe.xls").Sheets(1)
  Rows("2:2").Select
  Selection.Delete Shift:=xlUp
  Range("A1").Select
End Sub
```

2. Teil: Beschriftungen ergänzen

In der Textdatei für die Aufträge des aktuellen Monats wird eine Beschriftung nur ein einziges Mal in einer Spalte eingetragen. Für die Weiterbearbeitung mit Excel müssen die Beschriftungen in jeder einzelnen Zeile vorhanden sein, sonst ist es nicht möglich, die Daten zu sortieren und zusammenzufassen. Daher werden im folgenden Makro die fehlenden Beschriftungen ergänzt. Zur Durchführung werden einige interessante sehr mächtige Tabellenfunktionen von EXCEL verwendet.

1. Tabelle *Monat* aktivieren, → **MAKRO AUFZEICHNEN,** Makroname: *BeschriftungEinfügen*
2. In der Mappe ***Hilfe.XLS*** im Tabellenblatt ***Monat*** Zelle A1 anklicken
3. → **BEARBEITEN→ GEHE ZU** Schaltfläche **INHALTE** im Dialogfeld *Inhalte auswählen* die Option **AKTUELLER BEREICH** aktivieren und **OK**. Excel wählt den aktuellen Bereich aus, also das Rechteck der Zellen inklusive der aktiven Zelle, der von leeren Zellen oder der Tabellenbegrenzung umgeben ist.
4. → **BEARBEITEN → GEHE ZU** Schaltfläche **INHALTE** und Option **Leerzellen** markieren und **OK**. Dadurch sind nur noch die leeren Zellen der Auswahl markiert.
   Jede leere Zelle muss nun den Wert der ersten nicht leeren Zelle darüber erhalten.
5. Geben Sie = ein und drücken Sie ↑ – wenn z.B. der Zellbezug D2 in Zelle D3 steht, so bedeutet dies "eine Zelle nach oben und gleichen Spalte" – und geben Sie **Ctrl + Return** ein, dadurch wird die Formel in sämtliche ausgewählte Zellen kopiert. Jede Zelle, welche die neue Formel enthält, zeigt nun auf die darüberliegende Zelle.
   → **BEARBEITEN→ GEHE ZU** Schaltfläche **INHALTE** im Dialogfeld ***Inhalte auswählen*** die Option **Aktueller Bereich** aktivieren und **OK** oder kurz **Ctrl + ⇧ + ***
   → **BEARBEITEN → KOPIEREN**
   → **BEARBEITEN → INHALTE EINFÜGEN → WERTE → OK**
   Durch Drücken der Taste Esc Kopiermodus beenden.
6. Zelle A1 anklicken
7. → **AUFZEICHNUNG BEENDEN**

Sehen Sie sich das entstandene VBA-Programm an und vergleichen Sie Ihr Makro mit dem unten abgebildeten:

```
Sub BeschriftungEinfügen()
  Range("A1").Select
  Selection.CurrentRegion.Select
  Selection.SpecialCells(xlCellTypeBlanks).Select
  Selection.FormulaR1C1 = "=R[-1]C"
  Selection.CurrentRegion.Select
  Selection.Copy
  Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, _
    SkipBlanks:=False, Transpose:=False
  Application.CutCopyMode = False
  Range("A1").Select
End Sub
```

3. Teil: Eine Spalte mit Datumsangaben hinzufügen

Wir erstellen zuerst ein Makro durch Aufzeichnen, welches das Datum ***Jan 98*** in jede Zeile an der ersten Position einfügt.

1. → **MAKRO AUFZEICHNEN,** Makroname: *DatumEinfügen*

2. Zelle A1 anklicken und → **EINFÜGEN** → **SPALTEN** um eine neue Spalte einzufügen

3. Schreiben Sie *Datum* in Zelle A1 und wechseln Sie zu Zelle A2
   Ctrl + ⇧ + * um den aktuellen Bereich auszuwählen

4. → **BEARBEITEN** → **GEHE ZU** Schaltfläche **INHALTE** und Option **Leerzellen** markieren und **OK**. Dadurch sind die Zellen, in welche das Datum eingefügt werden soll ausgewählt.

5. Geben Sie ***Jan 98*** ein und **Ctrl** + **Return**. Dadurch wird überall das benötigte Datum eingefügt.

6. Zelle A1 anklicken

7. → **AUFZEICHNUNG BEENDEN**

Testen Sie das Makro und sehen Sie sich den Programmcode an.

Dieses Makro ist aber nur für den Monat Januar 1998 richtig. Im Februar müsste es entsprechend abgeändert werden. Aus diesem Grund ändern wir das Makro so ab, dass es während der Ausführung als erstes nach dem aktuellen Datum fragt. Wir verwenden dafür die Visual Basic-Funktion **`InputBox`**, die während der Ausführung eines Makros nach Informationen fragt. Der eingegebene Wert wird in einer Variablen mit der Bezeichnung ***neuDatum*** gespeichert.

Ändern Sie daher Ihr Programm entsprechend ab! Die neuen Anweisungen sind in der Programmliste markiert dargestellt:

```
Sub DatumEinfügen()
  Range("A1").Select
  Selection.EntireColumn.Insert
  ActiveCell.FormulaR1C1 = "Datum"
  Range("A2").Select
  neuDatum = InputBox("Geben Sie das Datum im Format mm-jj z.B. 1-98 ein!")
  Selection.CurrentRegion.Select
  Selection.SpecialCells(xlCellTypeBlanks).Select
  Selection.FormulaR1C1 = neuDatum
  Range("A1").Select
End Sub
```

4. Teil: Daten des neuen Monats an die bestehenden Daten anfügen

Die Daten des Tabellenblattes ***Monat*** sollen an die bestehende Excel-Tabelle mit dem Namen ***Aufträge.xls*** angehängt werden.

Zuerst werden die Daten des Tabellenblattes mit Ausnahme der Überschriften kopiert, dann wird *Aufträge.XLS* geöffnet, die erste leere Zelle wird ausgewählt und die Daten des neuen Monats werden dort eingefügt. Wir bezeichnen den gesamten Datenbestand der Tabelle mit dem Namen *Datenbank*.

1. Tabelle *Monat*, die bereits die Beschriftungen inklusive Datum enthält aktivieren und → **MAKRO AUFZEICHNEN,** Makroname: *DatenbankAnhängen*
2. Titelzeile löschen und den aktuellen Bereich durch **Ctrl** + ⇧ + * auswählen
3. → **BEARBEITEN** → **KOPIEREN**
4. → **DATEI** → **ÖFFNEN** Dateiname: **Aufträge.XLS** und öffnen
5. Mit **Ctrl** + **Home** Zelle A1 aktivieren und mit **Ctrl** + ↓ gelangen wir zur letzten Zeile der Datenbank, mit ↓ gelangen wir in die erste freien Zeile der Spalte A.
6. → **BEARBEITEN** → **EINFÜGEN** und durch Drücken der Taste ESC Meldungen der Statuszeile entfernen
7. Mit **Ctrl** + ⇧ + * wird der gesamte neue Datenbankbereich markiert, also einschliesslich der eben angehängten Zeilen.
8. → **EINFÜGEN** → **NAMEN** → **FESTLEGEN** und als Name der Arbeitsmappe ***Datenbank*** angeben und **OK**. Geben Sie bei Namen festlegen den Namen ein, wählen Sie keinen aus der Liste!
9. → **DATEI** → **SCHLIESSEN** Klicken Sie Nein an, wenn Sie gefragt werden, ob Änderungen gespeichert werden sollen, da wir zuerst dieses Makro testen wollen.
10. → **AUFZEICHNUNG BEENDEN**

Sehen Sie sich den Programmcode an.

In der vorhandenen Form funktioniert das Makro nur, wenn es unter den gleichen Bedingungen wie bei der Aufzeichnung ausgeführt wird. Im nächsten Monat – also im Februar- würde es nicht funktionieren. Denn durch die Anweisung `Range("A3333").Select` werden die Tabellenwerte vom nächsten Monat immer bei der absoluten Zelle A3333 eingefügt.

Wir müssen aber nach der ersten leeren Zelle am Ende der Datenbank suchen, diese Zelle liegt 1 Zeile unterhalb der letzten Zeile der Datenbank, d.h. wir müssen eine Zeile hinuntergehen relativ zum Ausgangspunkt.

Ersetzen Sie daher die obige Anweisung `Range("A3333").Select` durch:

```
ActiveCell.Offset(1, 0).Range("A1").Select
```

das bedeutet soviel wie: "wähle die Zelle unter der aktiven Zelle". Der neue ausgewählte Bereich besteht nur aus einer einzigen Zelle. Der neue Bereich wird so betrachtet als wäre er in der linken oberen Ecke einer vollkommen "virtuellen" Tabelle positioniert und es wird die Zelle A1 dieser "virtuellen" Tabelle ausgewählt.

Ersetzen Sie ausserdem die Zeile nach `Selection.CurrentRegion.Select` durch:

```
Selection.Name = "Datenbank"
```

Diese Anweisung weist dem Bereich, den die Datenbank am Monatsende einnimmt den Namen **Datenbank** zu.

Die Anweisung `ActiveWorkbook.Close` bewirkt, dass eine Eingabeaufforderung eingeblendet wird, die Sie fragt, ob Änderungen gespeichert werden sollen. Manchmal wissen Sie, dass Sie Änderungen immer speichern wollen (***True***) oder auch dass Sie niemals Änderungen speichern wollen (***False***). Mit folgender Anweisung können Sie den Prozess automatisieren.

Ersetzen Sie die vorhandene Close-Anweisung während der Testphase durch:

```
ActiveWorkbook.Close SaveChanges:=False
```

Wenn Sie den Test beendet haben, ändern Sie **`False` in `True`.**

Ihr VBA-Programm sollte nun so aussehen:

```
Sub DatenbankAnhängen()
  Range("A1").Select
  Selection.EntireRow.Delete
  Selection.CurrentRegion.Select
  Selection.Copy
  Workbooks.Open FileName:=".................\Aufträge.xls"
  Range("A1").Select
  Selection.End(xlDown).Select
  ActiveCell.Offset(1, 0).Range("A1").Select
  ActiveSheet.Paste
  Application.CutCopyMode = False
  Selection.CurrentRegion.Select
  Selection.Name = "Datenbank"
  ActiveWorkbook.Close SaveChanges:=True
  Range("A1").Select
End Sub
```

5. Teil: Die Tabelle löschen

Nach der Übernahme der Daten in die Datenbank, benötigen wir die importierte Tabelle nicht mehr. Daher löschen wir diese.

1. Die Tabelle *Monat* aktivieren
   → **MAKRO AUFZEICHNEN,** Makroname: *TabelleLöschen*

2. → **BEARBEITEN** → **BLATT LÖSCHEN** → **OK**

3. Aufzeichnung beenden.

Durch Einfügen der Anweisung `Application.DisplayAlerts = False` können Sie die Frage von EXCEL, ob Sie wirklich löschen wollen, unterdrücken.

Wird die Eigenschaft `DisplayAlerts` auf `False` gesetzt, so wird angenommen, dass Sie in allen Eingabeaufforderungen, die normalerweise eingeblendet würden, die voreingestellte Antwort wählen. Das gilt aber nur für die Ausführungszeit dieses Makros.

```
Sub TabelleLöschen()
  Application.DisplayAlerts = False
  ActiveWindow.SelectedSheets.Delete
End Sub
```

6. Teil: Alle Teile zusammenfügen

Der einfachste Weg Makros zusammenzuführen, besteht darin, ein Makro aufzuzeichnen, das andere Makros ausführt.

1. → **MAKRO AUFZEICHNEN,** Makroname: *MonatlichesProjekt*

2. → **MAKRO AUSFÜHREN,** Makroname: *DateiImportieren* → **AUSFÜHREN**
   Wählen Sie die Textdatei, die importiert werden soll und → **ÖFFNEN**

3. → **MAKRO AUSFÜHREN,** Makroname: *BeschriftungEinfügen* → **AUSFÜHREN**

4. → **MAKRO AUSFÜHREN,** Makroname: *DatumEinfügen* → **AUSFÜHREN**
   Geben Sie ein passendes Datum ein → **OK**

5. → **MAKRO AUSFÜHREN,** Makroname: *DatenbankAnhängen* → **AUSFÜHREN**

6. → **MAKRO AUSFÜHREN,** Makroname: *TabelleLöschen* → **AUSFÜHREN**

7. Aufzeichnung beenden

Sehen Sie sich den letzten Makro an und vereinfachen Sie ihn wie folgt:

```
Sub MonatlichesProjekt()
  DateiImportieren
  BeschriftungEinfügen
  DatumEinfügen
  DatenbankAnhängen
  TabelleLöschen
End Sub
```

Dialogfelder

Dialogfelder sind Standardelemente eines jeden Windows-Programms. Für die Kommunikation zwischen Anwender und Programm werden Fenster verwendet. Einer der Gründe für die Beliebtheit der Sprache Visual Basic ist die Möglichkeit, auf einfache Art benutzerdefinierte Fenster mit Dialogfeldern zu erstellen.

Es gibt bestimmte Fenster, die immer gleich aussehen, egal ob man sie von EXCEL oder von WORD aus aufruft, z.B. zum Öffnen und Schliessen von Dateien. Diese Fenster sind in Windows definiert. Die Programme greifen nur auf sie zurück. Solche Fenster werden in VBA **integrierte Dialogfenster** genannt und können auch von einem VBA-Programm aus angesprochen werden. Bis jetzt haben wir aus dieser Kategorie die Dialogfenster für Standardein- / -ausgabe verwendet: **InputBox** und **MsgBox**. VBA erlaubt auch das Erstellen von **benutzerdefinierten Dialogfeldern.** Für diese werden alle Standardelemente zur Verfügung gestellt, die von der Windowsoberfläche her bekannt sind: Schaltflächen, Optionsfelder, Kontrollkästchen, Textfelder, Listenfelder, Eingabefelder etc. In VBA ist ein speziell dafür vorgesehener Editor integriert, der den Entwurf der Dialoge weitgehend auf graphischem Weg ermöglicht.

Statt der Bezeichnung Dialogfeld wird häufig der Begriff Formular (UserForm) bzw. Eingabeformular verwendet, da Dialogfelder häufig für die Eingabe von Daten in ein Programm verwendet werden. Es ist dabei das Ziel, die Eingabe so zu gestalten, dass man nur die unbedingt notwendigen Angaben von Hand eintippt und bestimmte Informationen aus vorgegebenen Listen auswählt, wozu man z.B. Drop-Down-Listenfelder verwendet.

1. Beispiel: Dialogfelder im Arbeitsblatt

Da die Verwendung von grafischen Benutzeroberflächen die Bedienung eines Programms erleichtert, zeigen wir an einem Beispiel, wie man Steuerelemente (auch ActiveX - Steuerelemente genannt) direkt in einem Arbeitsblatt einfügen kann. Diese Kenntnisse sind bei der Erstellung von benutzerdefinierten Formularen in VBA sehr nützlich.

In diesem Beispiel werden wir folgende Steuerelemente verwenden:

Drehen-Schaltfläche oder Drehfeld (SpinButton) in zwei Ausführungen

Bildlaufleiste (ScrollBar)

Kombinationsfeld–DropDownListenfeld (ComboBox)

Erstellung eines Modells zur Berechnung von Darlehensraten

1. Starten Sie EXCEL und verwenden Sie eine neue Arbeitsmappe

2. Geben Sie in B2 bis B7 die Texte ein: *Preis, Anzahlung, Kredit, Zins, Laufzeit, Rate*
   Geben Sie in C2 ***5000*** ein, in C3 ***20%***, in C5 ***8%*** und in C6 ***3*** ein
   Geben Sie in C4 ein: ***=Preis *(1-Anzahlung)***
   Trickkiste von Excel
   EXCEL ermittelt die Zellen an Hand der Beschriftungen **neben den Zellen**

   Sehen Sie sich zur Kontrolle den Inhalt von C4 an:
   wählen Sie C4 aus und klicken Sie auf die Bearbeitungszeile. In der Formel wird das Wort **Preis** blau angezeigt und im Arbeitsblatt erscheint ein blauer Rahmen um die Zelle C2, die den Wert für Preis enthält. Das Wort **Anzahlung** wird grün angezeigt und im Arbeitsblatt erscheint ein grüner Rahmen um die Zelle C3.

Geben Sie in die Zelle C7 ein: =RMZ(Zins/12;Laufzeit*12;Kredit) oder verwenden Sie den Funktionsassistenten. Damit haben wir die monatliche Ratenzahlung berechnet.

Die Funktion **RMZ (R**egel**m**ässige **Z**ahlung**)** liefert die konstante Zahlung einer Annuität pro Periode, wobei konstante Zahlungen und ein konstanter Zinssatz vorausgesetzt werden. Nähere Angaben siehe Hilfe!

Bei Änderungen der Eingabegrössen wird automatisch die neue Rate berechnet. Durch Verwendung von verschiedenen Befehlsschaltflächen wollen wir im folgenden Teil die Eingabemöglichkeiten auf bestimmte sinnvolle Bereiche einschränken.

3. ***Laufzeit zwischen 1 bis 6 Jahre – nur ganze Zahlen***
Verwendung einer **Drehen – Schaltfläche** (SpinButton**)**

→ ANSICHT → SYMBOLLEISTEN → STEUERELEMENT-TOOLBOX
die **Drehen-Schaltfläche** anklicken. Halten Sie die Alt-Taste gedrückt und klicken Sie in die obere linke Ecke der Zelle E6. Wenn Sie dabei die Alt-Taste gedrückt halten, wird das Steuerelement am Zellengitter ausgerichtet. Lassen Sie die Alt-Taste los und ziehen Sie die untere rechte Ecke der neuen Drehen-Schaltfläche in die Mitte der unteren Randlinie von Zelle E6, dadurch wird die Schaltfläche gedreht und passt genau in die Zelle.

→ STEUERELEMENT-TOOLBOX → EIGENSCHAFTENFENSTER

Beachten Sie, dass die Drehen-Schaltfläche im Eigenschaftenfenster durch SpinButton bezeichnet wird.

Geben Sie **1** als Wert der Eigenschaft *Min*, **6** als Wert der Eigenschaft *Max* ein und **C6** bei der Eigenschaft *Linked Cell* und Return.

Jedes ActiveX-Steuerelement verfügt über verschiedene Eigenschaften. Bei vielen Eigenschaften sind Standardwerte vorgegeben, die man einfach übernehmen kann. Es müssen nur die Eigenschaften, für die benutzerdefinierte Werte benötigt werden, geändert werden.

→ STEUERELEMENT-TOOLBOX → ENTWURFSMODUS BEENDEN

Klicken Sie auf die Drehen-Schaltfläche im Tabellenblatt, um sie zu testen.

4. ***Anzahlung zwischen 0 und 100% mit Schrittweite 5%***
Verwendung einer **Drehen – Schaltfläche** (SpinButton)

Obwohl die Anzahlung als Prozentsatz durch eine Dezimalzahl angegeben wird, können Sie auch hier wie bei Nr.3 eine Drehen-Schaltfläche verwenden, sofern Sie die Zwischenergebnisse in einer eigenen Zelle speichern.

→ STEUERELEMENT-TOOLBOX → ENTWURFSMODUS
Kopieren sie die Drehen-Schaltfläche von Zelle E6 in die Zelle E3. Durch das Kopieren des Steuerelements wird sichergestellt, dass beide Steuerelemente gleich gross sind.

Geben Sie im Eigenschaftenfenster ein:
Für *Max* 100, für Min 0, für *LinkedCell* die Zelle **H3** und bei *SmallChange* 5. Mit SmallChange wird die Schrittweite gesteuert, um den der Wert bei jedem Klicken auf das Steuerelement geändert wird.

→ STEUERELEMENT-TOOLBOX → ENTWURFSMODUS BEENDEN

Geben Sie in Zelle C3 ein: =H3/100

Testen Sie anschliessend die Drehen-Schaltfläche für Anzahlung.

5. ***Zinssatz zwischen 0 und 20% mit Schrittweite 0.25%***
Verwenden einer **Bildlaufleiste** (ScrollBar)

Da die Bildlaufleiste genau so wie die Drehen-Schaltfläche nur ganzzahlige Werte zurückgibt, müssen wir auch dieses Steuerelement mit einer Zelle für das Zwischenergebnis verknüpfen.

→ **STEUERELEMENT-TOOLBOX** → **BILDLAUFLEISTE**

Halten Sie die Taste Alt gedrückt, während Sie in die linke obere Ecke von Zelle E5 klicken. Halten Sie die Taste Alt weiterhin gedrückt, während Sie die untere rechte Ecke der neu erstellten Bildlaufleiste in die untere rechte Ecke von Zelle E5 ziehen.

Geben Sie im Eigenschaftenfenster ein:
Für *Max* 2000, für *SmallChange* 25, für *LargeChange* 100 und für *LinkedCell* den Zellbezug **H5** und Return

→ **STEUERELEMENT-TOOLBOX** → **ENTWURFSMODUS BEENDEN**

Probieren Sie das Bildlaufleisten-Steuerelement aus, indem sie auf die Pfeilschaltflächen und in den dazwischenliegenden Bereich klicken. Wenn Sie auf eine der Pfeilschaltflächen klicken, wird der Wert in Zelle H5 in Schritten von 25 geändert. Wenn Sie in den Bereich zwischen den Pfeilschaltflächen klicken, wird der Wert in Schritten von jeweils 100 geändert.

Wählen sie die Zelle C5 aus und geben Sie ein: = H5/10000 und testen Sie die Schaltfläche.

6. ***Preis des Autos in Abhängigkeit des Modells aus einer vorher definierten Liste von gewünschten Gebrauchtautos mit Preisen auswählen***
Verwenden eines **Kombinationsfeldes – Dropdown-Listenfeld** (ComboBox)

Auswahlliste vorbereiten:
Geben Sie in die Zellen K2 bis K8 und L2 bis L8 die folgenden Informationen ein:

91 Mercury Sable	10'500.00	sFr
88 Nissan Pulsar NX	6'350.00	sFr
90 Toyota Camry	8'950.00	sFr
88 Dodge Lancer ES	6'299.00	sFr
87 BMW 325	7'959.00	sFr
91 Chev Camaro	6'796.00	sFr
88 Mazda MX6	8'500.00	sFr

Wählen Sie K2 und **Ctrl** + ⇧ + * (bewirkt die Markierung des eingegebenen Bereichs)
→ **EINFÜGEN** → **NAMEN** → **DEFINIEREN** Name: **Autoliste** → **OK**

→ **STEUERELEMENT-TOOLBOX** → **KOMBINATIONSFELD**

Halten Sie die Taste Alt gedrückt und ziehen Sie ein Rechteck von der oberen linken Ecke zur unteren rechten Ecke der Zelle E2.

Es gibt zwei Arten von Kombinationsfeldern:

- Dropdown – Listenfelder, in denen nur ein Eintrag aus einer Liste ausgewählt werden kann
- Listenfelder, die mit einem Bearbeitungsfeld kombiniert sind und in denen ein neuer Wert eingegeben oder ein Eintrag aus einer Liste ausgewählt werden kann.

Wir verwenden hier ein **Dropdown-Listenfeld**.

Geben Sie im Eigenschaftenfenster ein:

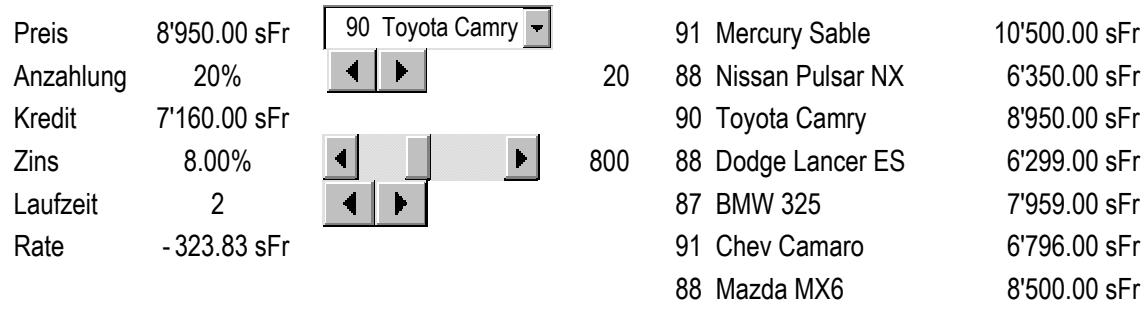
- bei der Eigenschaft *Style* den Eintrag 2 - **fmStyleDropDownList**
- bei *LinkedCells* den Wert **C2** und bei *ListFillRange* **Autoliste** und Return.

Die Zelle C2 ist mit dem Kombinationsfeld verknüpft und der Wert in C2 ändert sich, wenn im Kombinationsfeld ein anderes Auto ausgewählt wird. Zunächst wird nur die Bezeichnung des Autos angezeigt und nicht der Preis. Wir müssen noch folgende Eigenschaften ändern:

- *ColumnCount* auf **2** setzen, damit wird angegeben, dass für ListFillRange zwei Spalten mit Werten existieren.
- *BoundColumn* auf **2** setzen, damit wird gesteuert, aus welcher Spalte die Werte zu übernehmen sind.
- Geben Sie noch bei der Eigenschaft für Spaltenbreite folgende Werte ein, damit Sie nicht im Kombinationsfeld unten die Bildlaufleiste erhalten:
  ColumnWidths **2,6cm; 1,25cm** (Breiten der beiden Spalten)

→ **STEUERELEMENT-TOOLBOX** → **ENTWURFSMODUS BEENDEN**

Testen Sie nun nochmals alle Schaltflächen und probieren Sie die verschiedenen Möglichkeiten aus.



Bemerkung: Steht für die gewünschte Art der Währungsanzeige kein benutzerdefiniertes Format zur Verfügung, können Sie es selbst erstellen:

→ **FORMAT** → **ZELLEN** → Kategorie: **Benutzerdefiniert,**
Registerkarte: **Zahlen**

Sie erstellen ein benutzerdefiniertes Währungsformat, indem Sie Formatcodes eingeben. Dabei müssen Sie die Eigenkreationen in Hochkommas als Text einschliessen z.B. #'##0.00 "sFr".

Benutzerdefinierte Formate werden mit der Arbeitsmappe gespeichert. Soll Microsoft Excel standardmässig ein bestimmtes Währungssymbol anzeigen, ändern Sie das Währungssymbol in den Ländereinstellungen der Systemsteuerung, bevor Sie Microsoft Excel starten.

2.Beispiel: Dialogfelder in VBE mit VBA-Programmen

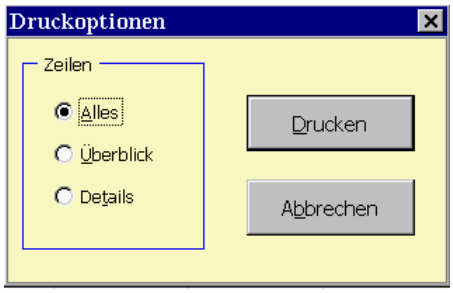
Wir werden ein benutzerdefiniertes Dialogfeld erstellen, um die Angaben der Budgettabelle in verschiedenen Versionen auszudrucken. Die Lösung dieser Aufgabe erfolgt in vier Schritten:

1.Teil: Entwurf der Benutzeroberfläche eines Dialogfeldes mit 3 eingerahmten Optionsfeldern und 2 Schaltflächen

2.Teil Budgettabelle für das Ausdrucken vorbereiten

3.Teil Erstellung der benötigten VBA-Programme

4.Teil Schaltfläche in der EXCEL-Menüleiste erstellen



1.Teil: ***Entwurf der Benutzeroberfläche eines Dialogfeldes mit 3 Optionsfeldern*** *(OptionsButtons)* ***und 2 Schaltflächen*** *(CommandButtons)*

1. Starten Sie Excel und öffnen Sie BUDGET.XLS und speichern Sie die Tabelle unter dem neuen Namen: VERSIONEN.XLS ab.

2. → **EXTRAS** → **VISUAL BASIC EDITOR** → **EINFÜGEN** → **USERFORM**

   Nach dem Einfügen erscheint ein neues Objekt mit dem Namen UserForm1 in der Liste des Projekt-Explorers. Normalerweise sollte auch die Symbolleiste **Werkzeugsammlung** erscheinen. Falls das nicht der Fall ist, fügen Sie diese z.B. mit Symbol ein *oder* → **ANSICHT** → **WERKZEUGSAMMLUNG**. Die Werkzeugsammlung erlaubt das Einfügen aller Fensterelemente.

3. Drücken Sie auf F5, um das Formular in der Excel-Tabelle anzuzeigen und schliessen Sie es anschliessend wieder durch Klicken auf die Schaltfläche FENSTER SCHLIESSEN in der oberen rechten Fensterecke.

4. Gestaltung des Dialogfeldes:

   → **ANSICHT**→ **EIGENSCHAFTENFENSTER**

   Alle benutzerdefinierbaren Änderungen im Dialogfenster lassen sich mit Hilfe des Eigenschaftenfensters durchführen. Die Eigenschaften lassen sich alphabetisch oder nach Kategorien geordnet anzeigen.

   - Ändern Sie **Caption** in ***Druckoptionen*** – dieser Text erscheint sofort in der Titelzeile des Dialogfeldes.
   - Ändern Sie **(Name)** in ***frmDrucken*** (frm steht für Formular). Wenn man in einer Prozedur das Formular verwenden will, muss man diesen Namen verwenden.
   - Im Eigenschaftenfenster können Sie unter der Kategorie **Darstellung** die Farben ändern: mit *Backcolor* die Hintergrundfarbe, mit *BorderColor* die Rahmenfarbe, wobei der Rahmen nur sichtbar ist, wenn *BorderStyle* auf 1 gesetzt ist, mit *ForeColor* die Farben der Texte. In der Kategorie **Schriftart**, können Sie Schriftart und Schriftgrösse über *Font* wählen. In der Gruppe **Position** können Sie mit *StartUpPosition* die Position am Bildschirm festlegen, in der ein Dialogfeld zur Laufzeit erscheint.

   Zu allen Eigenschaften gibt es eine Online-Hilfe-Seite, die meistens sehr ausführliche Erklärungen enthält. Ein Klick auf den Eigenschaftsnamen und das Drücken der F1 Taste liefert Ihnen die Informationen.

5. Hinzufügen von **3 Optionsfeldern mit einem Rahmen**:

   Klicken Sie auf das Formularfenster.

   Klicken Sie in der Werkzeugsammlung auf die Schaltfläche **RAHMEN** und erstellen Sie den Rahmen durch Ziehen der Maus.

   Klicken Sie in der Werkzeugsammlung auf die Schaltfläche **OPTIONSFELD** und klicken Sie dann in den zuerst erstellten Rahmen. Fügen Sie unterhalb zwei weitere Optionsfelder ein.

   Wenn Sie mehrere Steuerelemente gleichen Typs in ein Formular einfügen möchten, können Sie in der Werkzeugsammlung auf die gewünschte Schaltfläche doppelklicken. Diese Schaltfläche bleibt solange aktiviert, bis Sie sie erneut anklicken.

6. Eigenschaftenfenster aktivieren und den eingefügten Elementen, die Sie mit Hilfe der DropdownListe unter der Titelzeile auswählen können, folgende Eigenschaften zuweisen:

Frame1	(Name)	grpZeilen		
	Caption	Zeilen		
OptionButton		**1**	**2**	**3**
	(Name)	optAlles	optÜberblick	optDetails
	Caption	Alles	Überblick	Details
	Accelerator	A	Ü	t
	Value	True	False	False

   Rahmen: Werden Optionsfelder umrahmt, so kann nur eine Option gesetzt werden. Sonst haben Rahmen nur dekorative Bedeutung.

   Die Eigenschaft ***Accelerator*** legt die Zugriffstaste für ein Steuerelement fest. Die Texte bei den Optionsfeldern haben jeweils einen unterstrichenen Buchstaben. Mit Hilfe von **Alt** + diesem Buchstaben kann die Schaltfläche aktiviert werden (falls die Maus kaputt ist).

   Mit der Zuweisung des Werts **True** an die Eigenschaft ***Value*** wird diese Option per Voreinstellung aktiviert.
   Optionsfelder können nur zwei Zustände annehmen:

   - Aktiviert: im runden Feld ist ein schwarzer Punkt – dies entspricht dem Zustand True, d.h. das Element wurde gesetzt
   - Nicht aktiviert: Das Feld ist leer und weiss – dies entspricht dem Wert False, d.h. das Element wurde absichtlich deaktiviert

7. Ausrichtung der Steuerelemente im Formular:

   wählen Sie alle drei Optionsfelder aus, indem Sie in den Bereich zwischen dem unteren Optionsfeld und dem Rahmen klicken und dann ein Rechteck ziehen, das die Optionsfelder und die Bezeichnungen umschliesst.

   → **FORMAT** → **VERTIKALER ABSTAND** → **ENTFERNEN**

   → **FORMAT** → **AUSRICHTEN** → **LINKS**

   → **FORMAT** → **GRÖSSE ANPASSEN**

8. Speichern Sie die Arbeitsmappe und drücken Sie F5, um zu sehen, wie die Optionsfelder aussehen und klicken Sie probeweise auf die Optionsfelder.

9. Hinzufügen von zwei Befehlsschaltflächen für **Drucken** und **Abbrechen**; diese beiden Schaltflächen werden in fast allen Dialogen gebraucht

   Klicken Sie auf das Formularfenster.

   Klicken Sie in der Werkzeugsammlung auf das Steuerelement **BEFEHLSSCHALTFLÄCHE** und klicken Sie dann im Formular rechts neben den Rahmen mit den Optionsfeldern. Fügen Sie anschliessend darunter noch eine zweite Befehlsschaltfläche ein.

10. Eigenschaftenfenster aktivieren und den neuen Schaltflächen folgende Eigenschaften zuweisen:

CommandButton	**1**	**2**
(Name)	cmdDrucken	cmdAbbrechen
Caption	Drucken	Abbrechen
Accelerator	D	b
Default	True	False
Cancel	False	True

   Mit der Eigenschaft ***Default*** wird festgelegt, ob die Schaltfläche die Standardbefehlsschaltfläche ist – das ist die Schaltfläche, die durch Drücken der Enter-Taste aktiviert wird. Ein Formular kann nur eine Standardschaltfläche enthalten.

   Indem Sie die Eigenschaft ***Cancel*** mit dem Wert True definieren, weisen Sie die Schaltfläche als Schaltfläche zum Abbrechen der Ausführung aus – das ist die Schaltfläche, die durch Drücken der Taste ESC aktiviert wird.

11. Festlegen der Ereignisbehandlungsroutinen für die beiden Schaltflächen:

   Schaltfläche Abbrechen: beim Klicken auf diese Schaltfläche soll das Fenster geschlossen werden:

   Doppelklicken Sie auf die Schaltfläche **ABBRECHEN**, um eine Ereignisbehandlungroutine namens **`cmdAbbrechen_Click`** zu erstellen. Geben Sie in den Rumpf der Prozedur die Anweisung **`Unload Me`** ein.

   Mit dem Befehl **`Unload`** wird das Formular aus dem Speicher gelöscht Das Schlüsselwort **`Me`** ist eine Kurzschreibweise, die anstelle des Namens des Dialogfeldes verwendet werden kann.
   Die Anweisung **`Unload Me`** weist VB an, das Formular, das das Steuerelement enthält, dessen Ereignisbehandlungsroutine im Augenblick ausgeführt wird, aus dem Speicher zu entfernen.

   Schaltfläche Drucken:

   Wählen Sie aus der Liste Objekt im oberen Bereich des Codefensters cmdDrucken, um eine neue Prozedur namens **`cmdDrucken_Click`** zu erstellen. Geben Sie in den Rumpf der Prozedur die beiden Anweisungen ein:

```
Unload Me
MsgBox "Ausdruck läuft"
```

   Mit der ersten Meldung wird das Formular gelöscht, mit der zweiten Anweisung wird ein Meldungsfenster eingeblendet, das später durch die Druckfunktion ersetzt wird.

12. Speichern Sie die Arbeitsmappe und führen Sie das Formular mehrmals aus. Klicken Sie abwechselnd auf die Schaltflächen Drucken und Abbrechen – und auch auf die Enter-Taste und die Taste ESC.

13. Aktivierreihenfolge der Steuerelemente festlegen.

    Führen Sie das Formular nochmals aus und drücken Sie dann wiederholt auf die Tabulatortaste. Beachten Sie wie der kleine graue Rahmen von Steuerelement zu Steuerelement wandert. Der Rahmen kennzeichnet das Steuerelement, das den Fokus besitzt. Die Steuerelemente dieses Formulars sollten folgende Aktivierreihenfolge ausweisen:

    optAlles, optÜberblick, optDetails, cmdDrucken und *cmdAbbrechen*

    Mit folgenden Anweisungen können Sie die Reihenfolge ändern:

    Klicken Sie auf Abbrechen, um das Formular zu schliessen. Markieren Sie in VBE das Formular. → **ANSICHT** → **AKTIVIERREIHENFOLGE,** mit ↑ bzw. ↓ können Sie die Position verschieben. Die im Rahmen angegebenen Steuerelemente sind in *grpZeilen* enthalten. Durch Markieren des Rahmens im Formular und Wiederholung der obigen Menübefehle können Sie auch die Reihenfolge der Optionsfelder ändern.

    Auch mit der Eigenschaft ***TabIndex*** könnten Sie der Reihe nach die Elemente mit 0, 1, 2 etc. bezeichnen und dadurch die Reihenfolge angeben.

> Dialog testen:
>
> In der Entwurfsphase kann man immer durch Verwendung von F5 zwischendurch das Dialogfeld testen – auch wenn noch keine Programmzeile geschrieben wurde.
>
> *Qickinfo* für Steuerelemente:
>
> Über die Eigenschaft ***ControlTipText*** kann man für jedes Steuerelement eine Erklärung definieren.

2.Teil Budgettabelle für das Ausdrucken vorbereiten

Wir benötigen drei verschiedene Versionen der Tabelle zum Ausdruck:

Gesamte Tabelle	***Alles***
Überblick über Budgetdaten ohne Details	***Überblick***
Nur Detaildaten ohne Überblick	***Details***

Wir erstellen daher drei verschiedene benutzerdefinierte Ansichten, die wir im Arbeitsblatt speichern und später bei Bedarf aktivieren können. Die verschiedenen Ansichten erhalten wir durch Ausblenden der nicht benötigten Zeilen und Spalten.

Erste Ansicht Alles

Aktivieren sie die EXCEL-Tabelle mit den Budgetdaten

→ **ANSICHT** → **ANSICHT ANPASSEN** → Benutzerdefinierte Ansichten → **HINZUFÜGEN**

Name ***Alles***

Kontrollkästchen Druckeinstellung deaktivieren, das andere Kontrollkästchen aktiviert lassen → **OK**

Zweite Ansicht Überblick

Zeilen mit Detaildaten ausblenden: die Spalte B enthält ausschliesslich Bezeichnungen für Detaildaten

Spalte B markieren → **BEARBEITEN** → **GEHE ZU** → **INHALTE**
Option ***Konstanten*** anklicken → **OK**
→ **FORMAT** → **ZEILE** → **AUSBLENDEN**

Spalte D markieren → **BEARBEITEN** → **GEHE ZU** → **INHALTE**
Option ***Leerzellen*** anklicken → **OK**
→ **FORMAT** → **ZEILE** → **AUSBLENDEN**

→ **ANSICHT** → **ANSICHT ANPASSEN** → Benutzerdefinierte Ansichten → **HINZUFÜGEN**
Name ***Überblick***
Kontrollkästchen Druckeinstellung deaktivieren, das andere Kontrollkästchen aktiviert lassen → **OK**

Dritte Ansicht Details

Alle Zeilen ausblenden in denen Daten zusammengefasst sind

Rufen Sie benutzerdefinierte Ansicht **Alles** auf, damit wieder alle Zeilen angezeigt werden. Markieren Sie den Bereich A4:A68, dieser enthält die Beschriftungen für die auszublendenden Zeilen

→ **BEARBEITEN** → **GEHE ZU** → **INHALTE** Option ***Konstanten*** anklicken → **OK**
→ **FORMAT** → **ZEILE** → **AUSBLENDEN**

Spalte D markieren → **BEARBEITEN** → **GEHE ZU** → **INHALTE**
Option ***Leerzellen*** anklicken → **OK**
→ **FORMAT** → **ZEILE** → **AUSBLENDEN**

→ **ANSICHT** → **ANSICHT ANPASSEN** → Benutzerdefinierte Ansichten → **HINZUFÜGEN**
Name ***Details***
Kontrollkästchen Druckeinstellung deaktivieren, das andere Kontrollkästchen aktiviert lassen → **OK**

Arbeitsmappe speichern und die drei Ansichten ausprobieren. Zuletzt **Alle.**

3.Teil Erstellung der benötigten VBA-Programme

VBA-Programm zum Hin- und Herschalten zwischen den Ansichten erstellen:

1. Aufzeichnung starten: Schaltfläche *Makro aufzeichnen* *oder*
   → **EXTRAS** → **MAKRO** → **AUFZEICHNEN ...** Name des Makros: **AnsichtAnzeigen**

2. Rufen Sie die Ansicht **Überblick** auf und beenden Sie nachher die Aufzeichnung
   Sie erhalten folgendes Makro:

```
Sub AnsichtAnzeigen()
  ActiveWorkbbok.CustomViews("Überblick").Show
End Sub
```

Arbeitsmappen verfügen über eine Auflistung – CustomViews; der Name der Ansicht wird zum Aufruf eines Elements dieser Auflistung verwendet. Dieses Element verfügt über eine Show-Methode. Um zwischen den Ansichten auszutauschen, müssen Sie den in Klammern angegebenen Namen der Ansicht austauschen. Statt drei verschiedene Makros zu erstellen, können Sie den Namen der gewünschten Ansicht als Argument übergeben:

Ersetzen Sie "Überblick" durch AnsichtName

```
Sub AnsichtAnzeigen(AnsichtName)
  ActiveWorkbook.CustomViews(AnsichtName).Show
End Sub
```

3. Testen Sie das Makro und die Argumente über das Direktfenster:

   → **ANSICHT** → **DIREKTFENSTER**

   Geben Sie **AnsichtAnzeigen "Details"** ein und **Enter** und probieren Sie auch die anderen Argumente aus.

4. Schliessen Sie das Direktfenster und speichern Sie die Arbeitsmappe.

VBA-Programm zur Bestimmung der ausgewählten Option und des Druckens

Der gesamte Programmcode wird in die Prozedur **`cmdDrucken_Click`** eingefügt.

1. Im VBE → **ANSICHT** → **PROJEKTEXPLORER**

2. Bei **FORMULARE** die UserForm ***frmDrucken*** aktivieren und **CODE ANZEIGEN**

3. Zum Feststellen, welches Optionsfeld ausgewählt ist, fügen wir in das Programm eine **For Each Schleife** ein und durchsuchen die Controls-Auflistung des Rahmen-Steuerelements nach dem Wert *True*.

   Fügen Sie folgenden Programmcode ein:

```
Dim neuOption As Control
Dim neuAnsicht
For Each neuOption In grpZeilen.Controls
  If neuOption.Value = True Then
    neuAnsicht = neuOption.Caption
  End If
Next neuOption
AnsichtAnzeigen neuAnsicht
```

   Die Schleife speichert die Beschriftung der ausgewählten Option in einer Variablen mit dem Namen `neuAnsicht`. Diese Variable wird später während der Ausführung des VBA-Programms *AnsichtAnzeigen* als Argument verwendet. Es zeigt sich erst nun, weshalb wir denselben Namen für die benutzerdefinierten Ansichten und die Bezeichnungen der Optionsfelder verwendet haben.

4. Wenn Sie die Durchführung des Programms überprüfen wollen, können Sie durch wiederholtes Drücken von F8 das Programm schrittweise ausführen und damit den Ablauf verfolgen.

   Speichern Sie die Arbeitsmappe.

Makro zum Anzeigen des Dialogfeldes erstellen

1. → **ANSICHT** → **PROJEKTEXPLORER**

   Doppelklick auf Modul1 – dieses Modul enthält bereits den Makro **AnsichtAnzeigen**

2. Fügen Sie folgendes Makro ein:

```
Sub FormularAnzeigen()
  frmDrucken.Show
End Sub
```

   Mit der Show Methode wird das Dialogfeld am Bildschirm angezeigt und es wird der Dialog in den Speicher geladen, falls er noch nicht geladen ist. Sie nehmen über den Namen, den Sie dem Dialogfeld zugewiesen haben, darauf Bezug.

3. Testen Sie das Programm direkt von VBE aus durch Drücken von F5 .

4.Teil Schaltfläche in der EXCEL-Menüleiste erstellen

Beim Öffnen der Arbeitsmappe soll automatisch die Schaltfläche **BERICHT DRUCKEN** in die Excel-Menüleiste eingefügt werden, beim Schliessen der Mappe soll diese Schaltfläche wieder entfernt werden.

1. Befehlsschaltfläche beim Öffnen der Mappe einblenden

   → **ANSICHT** → **PROJEKTEXPLORER**

   Doppelklick auf das Objekt *DieseArbeitsmappe*. Wählen Sie *Workbook* aus der Objektliste und fügen Sie die folgenden Anweisungen in den Rumpf der Prozedur **Workbook_Open** ein:

```
Dim neuSchaltfläche As CommandBarButton
Set neuSchaltfläche = _
  Application.CommandBars("Worksheet Menu Bar").Controls.Add
neuSchaltfläche.Caption = "Bericht drucken"
neuSchaltfläche.Style = msoButtonCaption
neuSchaltfläche.BeginGroup = True
neuSchaltfläche.OnAction = "FormularAnzeigen"
```

   Mit der Add-Methode wird die Schaltfläche – das CommandBarButton-Objekt – in die Menüleiste gebracht. Die Beschriftung der Schaltfläche erfolgt mit `Caption`, durch `Style` wird im Menü die Beschriftung statt einem Symbol angezeigt, mit `BeginGroup` wird ein Trennstrich links neben dem Befehl eingefügt, um diesen von den Standardbefehle zu unterscheiden. Bei `OnAction` wird der Name des Makros zugewiesen, der ausgeführt werden soll.

2. Befehlsschaltfläche beim Schliessen der Mappe löschen

   Wählen Sie aus der Liste Prozedur im Bereich des Codefenster BeforeClose. Fügen Sie folgende Anweisungen als Körper der Prozedur **Workbook_BeforeClose** ein:

```
ActiveWorkbook.Save
On Error Resume Next
Application.CommandBars("Worksheet Menu Bar") _
  .Controls("Bericht drucken").Delete
```

Mit der ersten Anweisung wird die Arbeitsmappe gespeichert. Damit wird verhindert, dass Excel fragt, ob Sie die Arbeitsmappe speichern wollen. Mit der zweiten Anweisung wird Visual Basic davon abgehalten eine Warnmeldung auszugeben, wenn der Befehl **Bericht drucken** aus irgendeinem Grund nicht vorhanden ist. Mit der dritten Anweisung wird der Menübefehl **Bericht drucken** gelöscht.

Diese Anweisungen werden immer dann ausgeführt, wenn die Arbeitsmappe geschlossen wird und "verwischen" alle Spuren des Befehls **Bericht drucken**.

Achten Sie darauf, dass diese Prozedur ohne Fehler ist und ausgeführt werden kann, sonst haben Sie den eingefügten Menübefehl in allen EXCEL-Mappen!

3. Wechseln Sie in die EXCEL-Mappe, speichern und schliessen Sie die Arbeitsmappe und öffnen Sie die Mappe anschliessend wieder, probieren Sie den Menübefehl **BERICHT DRUCKEN** aus und kontrollieren Sie, ob dieser Befehl beim Schliessen der Mappe verschwindet.

Ausführung von erstellten VBA-Programmen

Die Ausführung von VBA-Programmen von VBE aus, also aus der Entwicklungsumgebung, ist etwas umständlich.

Es bestehen folgende Möglichkeiten des Starts eigener Programme:

- **Schaltfläche in Tabelle**
  wenn ein Makro nur in der Mappe, in der es gespeichert ist, verwendet werden soll, kann es mit einer Schaltfläche im Tabellenblatt oder auch im Diagrammblatt verbunden werden (siehe Beispiel 5 in Kapitel2)
- **Tastenkombination (Shortcut)**
  kann bei der Aufzeichnung bereits angegeben werden oder später über → **EXTRAS** → **MAKRO** → Name des Makros markieren → **MAKRO-OPTIONEN ...** In diesem Fall genügt Öffnen der EXCEL-Mappe und Ctrl + Buchstabe

  Im Kapitel 1 haben wir das bei den ersten Beispielen bereits kennengelernt
- **Schaltfläche in vorhandener oder eigener Symbolleiste**
- **Schaltfläche in vorhandener oder eigener Menüleiste**

Eigene Symbolleisten zum Starten der VBA-Programme

Excel besitzt verschiedene Symbolleisten. Im allgemeinen sind die Leisten **Standard** und **Format** aktiv. Über → **ANSICHT** → **SYMBOLLEISTEN** → **ANPASSEN** können Sie neue Symbolleisten und Schaltflächen erstellen.

Wir zeigen als Beispiel die Erstellung einer eigenen Symbolleiste, in der wir unsere eigenen Symbole zum Aufruf folgender VBA-Programme einfügen werden:

- Subprozedur aufrufen ***Zahlenlotto_ch***
- Funktion aufrufen ***Celsius / Fahrenheit***
- Dialogfeld anzeigen ***DialogfeldAnzeigen***
- Bild anzeigen ***BildAnzeigen***
- Klang erzeugen ***Sound1***

Die Symbolleiste könnte folgendermassen aussehen:

Öffnen Sie in EXCEL Ihre Mappe mit dem Namen: ***Meine_Mappe.XLS***

1. Erstellen einer benutzerdefinierten Symbolleiste

→ **ANSICHT** → **SYMBOLLEISTEN** → **ANPASSEN** Register Symbolleisten → **NEU**
Geben Sie als Name für die neue Symbolleiste z.B. ***LUCIA*** ein. Sobald Sie das Fenster für die Benennung schliessen, erscheint eine kleine noch leere Symbolleiste, in diese wollen wir eigene Schaltflächen zu Steuerung unserer Programme einfügen.

2. Neue Schaltflächen in die neue Symbolleiste einfügen

Öffnen Sie das Fenster **ANPASSEN** und aktivieren Sie das Register ***Befehle***. Markieren Sie die Kategorie ***Makros***. Ziehen Sie den Eintrag ***Benutzerdefinierte Schaltfläche*** aus der Befehlsliste bei gedrückter linker Maustaste in Ihre eigene Symbolleiste ***Lucia***.
Die Einfügestelle wird mit einem kleinen senkrechten Balken markiert. Beim Ziehen des Eintrags in die Symbolleiste wird die Schaltfläche **AUSWAHL ÄNDERN** aktiviert. Klicken Sie darauf und drücken Sie **a**, um das Feld **Name** auszuwählen.
Geben Sie **Lotto&zahlen** als neuen Namen ein – das Zeichen **&** hat die gleiche Funktion wie die Eigenschaft ***Accelerator*** im Eigenschaften-Fenster. Der Eintrag im Feld Name legt fest, welche Quickinfo für die Schaltfläche angezeigt wird. Der Eintrag **STANDARD** ist aktiv. Wählen Sie den Eintrag **SCHALTFLÄCHENSYMBOL UND TEXT**. Dadurch erhält die Schaltfläche Text und ein Symbol. Das Symbol editieren wir später.

3. VBA-Programm *Zahlenlotto_ch* einer neuen Schaltfläche zuweisen

Gehen Sie im Fenster **ANPASSEN** auf das Register ***Befehle***. Aktivieren Sie die neue Schaltfläche durch Anklicken und gehen Sie im Fenster **ANPASSEN** auf die Schaltfläche **AUSWAHL ÄNDERN**. Es öffnet sich ein Menü, dessen letzter Eintrag ***MAKRO zuweisen*** ist. Wenn Sie diesen Eintrag anklicken, öffnet sich ein Fenster mit der Liste der momentan zur Verfügung stehenden Makros. Sie müssen nur den Namen des Makros: **Zahlenlotto_ch** wählen und **OK**. Damit ist die Verbindung zwischen Schaltfläche und Makro erstellt, d.h. ab sofort führt ein Klick auf die Schaltfläche die angegebene Prozedur aus.
Probieren Sie die neue Schaltfläche aus, lassen Sie sich auch die QuickInfo anzeigen.

4. Editieren der neuen Schaltflächen

Das Editieren einer Schaltfläche setzt ein geöffnetes Fenster **ANPASSEN** voraus.
Wenn Sie mit der rechten Maustaste auf eine beliebige Symbolleiste klicken, erhalten Sie über das Kontextmenü sofort den Befehl Anpassen.
Alle für die Editierung wichtigen Einträge sind im Register ***Befehle*** über die Schaltfläche **AUSWAHL ÄNDERN** zu finden *oder* über das Kontextmenü auf dem Symbol.

Schaltflächensymbol ändern

Dieser Eintrag öffnet eine Liste von neuen Schaltflächen, die in Excel nicht verwendet werden. Wählen Sie sich daraus ein schönes Symbol aus. Dieses können sie anschliessend mit dem Schaltflächen - Editor bearbeiten.

Schaltflächensymbol bearbeiten

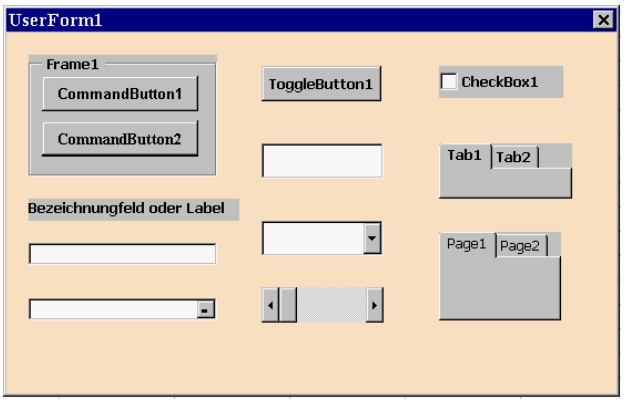
Excel stellt einen **Schaltflächen - Editor** zur Verfügung. Der Editor zeigt das Symbol in einem Raster und bietet 16 Farben für die Bearbeitung an. Man braucht nur auf eine der Farben klicken, um sie auszuwählen und dann kann man die einzelnen Pixel anklicken, um sie zu färben. Wenn ein schon gefärbtes Pixel nochmals angeklickt wird, bekommt es die graue, schattierte Hintergrundfarbe. Gestalten Sie Ihr Symbol nach Ihren Wünschen → **OK**

Mit **SCHALTFLÄCHENSYMBOL KOPIEREN** können Sie ein Bild aus einer Schaltfläche in die Zwischenablage kopieren, diese über ein Graphikprogramm z.B. Programm **Paint (→ START → PROGRAMME → ZUBEHÖR)** bearbeiten – aber auch hier stehen Ihnen zum Aufbau des Bildes nur 15 x 16 Pixel zur Verfügung – und zurück in die Zwischenablage kopieren. Von dort können Sie das neue Symbol mit **SCHALTFLÄCHENSYMBOL EINFÜGEN** wieder in die Symbolleiste übernehmen. Mit den erwähnten Schaltflächen können Sie auch vorhandene Symbole von Symbolleisten (z.B. aus der Steuerelement-Toolbox) in ihre eigene Symbolleiste übertragen.

5. ***VBA-Funktion*** zur Umrechnung ***Fahrenheit in Celsius*** einer Schaltfläche zuweisen

Es können nur Sub-Prozeduren, die sich in allgemeinen Modulen einer Mappe befinden direkt an Schaltflächen angebunden werden. Benutzerdefinierte Funktionen müssen daher über ein sogenanntes Hauptprogramm – das aber auch mit **`Sub`** beginnt – aufgerufen werden.
Erstellen Sie eine neue Schaltfläche mit dem Text ***Fahrenheit*** und einem dazu passenden Symbol in Ihrer eigenen Symbolleiste und verbinden Sie diese mit dem VBA-Programm: ***Main_Temperatur*** .

6. ***VBA-Dialogfeld*** einer Schaltfläche zuweisen



Erstellen Sie zur Illustration eine Userform und fügen Sie alle möglichen Steuerelemente ein. Dieses Beispiel soll nur den Aufruf eines Dialogfeldes aus einer Symbolleiste zeigen und hat sonst keine Bedeutung.

Es können nur Sub-Prozeduren, die sich in allgemeinen Modulen einer Mappe befinden direkt an Schaltflächen angebunden werden. Um die Anzeige von Dialogfeldern, die durch Prozeduren in der Userform definiert werden, über eine Schaltfläche zu starten, müssen wir ein neues Modul einfügen.

Fügen Sie ein neues Modul ein und geben Sie ein:

```
Sub Dialogfeldanzeigen()
  UserForm1.Show
End Sub
```

Vor dem Schlüsselwort *Show* ist der Name der Userform, also bei uns *Userform1* einzutragen.

Erstellen Sie eine neue Schaltfläche mit dem Text ***Dialogfeld Test*** und einem dazu passenden Symbole in Ihrer eigenen Symbolleiste und verbinden Sie diese mit dem VBA-Programm: ***Dialogfeldanzeigen***.

7. Anzeige eines Bildes einer Schaltfläche zuweisen

Fügen Sie eine neue Userform ein. Als Name wird automatisch *Userform2* gewählt. Durch Einfügen des Steuerelementes **Anzeige** (Image) ist das Einbinden von Graphik in Dialogfelder möglich.

Im Eigenschaftenfenster klicken wir auf die Eigenschaft ***Picture***. Es erscheinen *3 Punkte*, ein Klick auf diese Schaltfläche öffnet ein Fenster für die Angabe der **Bilddatei.** Es werden folgende Formate akzeptiert: **.bmp, *.cur, *.gif, *.ico, *.jpg* und **.wmf*. Wählen Sie aus. Mit ***PicureAlignment*** können Sie die Position des Bildes im Steuerelement definieren, falls der Umriss der Anzeige grösser als das Bild ist. ***PictureSizeMode*** dient zur Anpassung der Grösse des Umrisses und des Bildes.

Geben sie in einem Modul ein:

```
Sub Bild()
  UserForm2.Show
End Sub
```

Erstellen Sie eine neue Schaltfläche mit dem Text ***Bild anzeigen*** und einem dazu passenden Symbole in Ihrer eigenen Symbolleiste und verbinden Sie diese mit dem VBA-Programm: ***Bild***.

Sie können in der `Userform2` ein **Bezeichnungsfeld** - Steuerelement (Label) als Erläuterung zu dem Bild einfügen und z.B. *"Bitte das Bild anklicken"* in Caption eingeben. Ausserdem können Sie auch eine Meldung anzeigen, wenn Sie das Bild anklicken. Dazu müssen Sie eine Prozedur für das Click-Ereignis schreiben wie z.B.:

```
Private Sub Image1_Click()
  MsgBox "Ich bin ein Bild – bitte schliessen!"
End Sub
```

8. Klang einer Schaltfläche zuweisen

Fügen Sie ein neues Modul ein und geben Sie folgenden Programmcode ein:

```
Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
(ByVal WAVDatei As String, ByVal Wiedergabemodus As Long) As Long

Function PlaySound(SoundDatei)
  If Dir$(SoundDatei) > "" Then
    sndPlaySound SoundDatei, 1
    PlaySound = SoundDatei
  Else
    PlaySound = "Datei nicht gefunden"
  End If
End Function

Sub Sound1()
  PlaySound "C:\winnt\media\chimes.wav"
End Sub
```

Von den vorhandenen Wave-Dateien haben wir zur Illustration `chimes.wav` ausgewählt.

Erstellen Sie eine neue Schaltfläche mit dem Text *Kling* und einem dazupassenden Symbol und verbinden Sie diese mit dem VBA-Programm **`sound1`**.

Benutzerdefinierte Symbolleisten sind von jeder beliebigen Mappe aus zugänglich und bleiben solange Bestandteil von EXCEL, bis Sie sie wieder entfernen.

9. Entfernen von eigenen Schaltflächen oder Symbolleisten

→ **ANSICHT** → **SYMBOLLEISTEN** → **ANPASSEN**

Schaltfläche entfernen: ziehen Sie die Schaltfläche, die aus der Symbolleiste entfernt werden soll, mit der Maus auf das Fenster **ANPASSEN**.

Symbolleiste entfernen: Markieren Sie die benutzerdefinierte Symbolleiste im Register **Symbolleisten** und klicken Sie auf → **LÖSCHEN**

10. Benutzerdefinierte Symbolleiste mit einer Mappe öffnen und schliessen

Symbolleiste beim Öffnen einer Mappe automatisch anzeigen:

Das automatische Öffnen wird über ein kleines VBA-Programm erreicht. Es muss in die Ereignisprozedur des Ereignisses ***Open*** der Arbeitsmappe geschrieben werden:

→ **PROJEKT-EXPLORER** Doppelklick auf ***DieseArbeitsmappe***, das Code-Fenster wird geöffnet und wir wählen aus dem linken Feld ***Allgemein*** den Eintrag ***Workbook***. Im rechten Feld des Code-Fensters wählen wir aus der Liste der Ereignisse das Ereignis ***Open*** aus und fügen in den vorhandenen Prozedurkopf eine Zeile ein:

```
Private Sub Workbook_Open()
  Application.CommandBars("Lucia").Visible = True
End Sub
```

Die Eigenschaft **`Visible`** der Symbolleiste muss auf **`True`** gesetzt werden.

Symbolleiste beim Schliessen einer Mappe automatisch entfernen:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
  Application.CommandBars("Lucia").Visible = False
End Sub
```

Bemerkung zu Menüs

In Office wurde die Trennung zwischen Schaltflächen und Menüs aufgehoben. In Menüs können Symbole erscheinen und in einer Symbolleiste können Menüeinträge erscheinen. Schaltflächen und Menüeinträge sind in Office gleichgesetzt.

Wir beschreiben nur die Erstellung eines neuen Eintrags für ein Menü:

Fenster **ANPASSEN** Register ***Befehle***. Als ***Kategorie*** letzten Eintrag auswählen: **NEUES MENÜ**. Diese Kategorie verfügt über den gleichnamigen Befehl ***Neues Menü***. Markieren Sie diesen Befehl *Neues Menü* und ziehen Sie ihn auf eine bestehende Menüleiste. Nach dem Einfügen lässt sich das neue benutzerdefinierte Menü durch Anklicken öffnen und es können beliebige Befehle oder Schaltflächen aus dem Vorrat des Fensters **ANPASSEN** durch Ziehen mit der Maus in das neue Menü eingefügt werden. Die Bearbeitung der Menübefehle und das Anbinden an eigene Programme erfolgt in gleicher Weise wie bei den Symbolleisten.

Erstellen eines benutzerdefinierten Dialogfeldes

1. Erstellen eines Dialogfeldes (einer UserForm)

VBE → EINFÜGEN → USERFORM

Statt Dialogfeld oder UserForm wird auch der Begriff Formular bzw. Eingabeformular verwendet. Wenn ein Projekt mehrere Dialogfelder umfasst, werden diese automatisch mit Userform1, Userform2 etc. benannt.
Die Grösse des Dialogfensters können Sie direkt in der UserForm bestimmen. Die Gestaltung des Dialogfensters erfolgt über das Eigenschaftenfenster:

→ ANSICHT → EIGENSCHAFTENFENSTER

Genaue Informationen über die einzelnen Eigenschaften können Sie in der Hilfe finden.

Klick auf Eigenschaft und Taste F1

Wir erwähnen im Folgenden einige Eigenschaften:

Caption	Dialogfeldtitel
(Name)	Name statt Userform1 – wichtig, falls Sie mehrere Dialogfelder haben, wählen Sie aussagekräftige Namen
StartupPosition	Position des Dialogfeldes am Bildschirm
BackColor	Hintergrundfarbe auswählen über Register System oder Palette
BorderStyle	falls auf 1 gesetzt, wird ein Rahmen sichtbar
BorderColor	Rahmenfarbe
Font	Schriftart und Schriftgrösse
SpecialEffect	falls auf 1 gesetzt, erfolgt Dialogfeld in Reliefdarstellung

2. Einfügen von Steuerelementen in eine UserForm

Suchen Sie in der Toolbox (Symbolleiste Werkzeugsammlung) das gewünschte Steuerelement und ziehen Sie es auf das Formular. Wenn Sie das Steuerelement durch Doppelklick aktivieren, können Sie dieses Element mehrmals nacheinander einfügen.

Mehrfachmarkierung:
Mehrere Elemente im Dialogfeld können bei gedrückter Shift-Taste oder Ctrl-Taste mit der Maus angeklickt und dadurch markiert werden. Bei der **Shift-Taste** werden alle Elemente markiert, die sich zwischen dem ersten und dem letzten angeklickten Objekt befinden – einen ähnlichen Effekt kann man auch erreichen, wenn man um die Elemente einen **Rahmen** zeichnet, dadurch werden alle Elemente in seinem Inneren markiert. Durch Verwendung der **Ctrl-Taste** werden nur die angeklickten Objekte markiert.

Wenn mehrere Objekte markiert sind, können die Objekte aneinander ausgerichtet werden. Unter dem **Menü FORMAT** findet man die dazugehörigen Möglichkeiten z.B. Ausrichten (links, zentriert, rechts, oben, mitte, unten, am Raster), Grösse angleichen, Grösse anpassen, Horizontaler Abstand, Vertikaler Abstand, Im Formular zentrieren, Schaltflächen ausrichten, Reihenfolge. Die Einträge sind von der Markierung abhängig. Wenn

mehrere Objekte markiert sind, kann man für sie einige Eigenschaften im Eigenschaftenfenster setzen z.B. alle Darstellungseigenschaften.

Bemerkung zur Auswahl der Steuerelemente für Ihre Dialogfelder:

Die Eingabe in ein Dialogfeld sollte so gestaltet werden, dass Informationen aus vorgegebenen Listen ausgewählt werden können und nur die unbedingt notwendigen Angaben von Hand eingetippt werden müssen.

3. Festlegen von Steuerelementseigenschaften

Klicken Sie im Entwurfsmodus mit der rechten Maustaste auf ein Steuerelement und klicken Sie auf Eigenschaften, um das Eigenschaftenfenster anzuzeigen.

Geben Sie bei ***(Name)*** einen aussagekräftigen Namen an, den Sie dann in Ihren Programmen verwenden können, legen Sie mit ***Caption*** die Beschriftung des Steuerelementes fest. Für die Details zur Editierung bezüglich Grösse, Farbe, Position etc. siehe Punkt 1.

Bei der Eigenschaft ***ControlTipText*** können sie einen Text eingeben, dieser wird dann als QuickInfo angezeigt, wenn Sie zur Laufzeit mit der Maus über das Steuerelement im Fenster darüberfahren.

Legen Sie die Aktivierungsreihenfolge der Steuerelemente im Dialogfeld fest. Die Elemente werden normalerweise in der Reihenfolge notiert, in der sie eingefügt worden sind. Über die Eigenschaft ***Tabindex*** kann dies geändert werden. Das erste Element bekommt den Wert 0, das bedeutet dieses Element hat beim Öffnen des Dialogs den Fokus. Wenn es sich um ein Feld handelt, das eine Benutzereingabe annehmen kann, blinkt der Cursor im Inneren des Elements. Sie können auch über → **ANSICHT** → **AKTIVIERREIHENFOLGE** die Reihenfolge festlegen.

Mit der Eigenschaft ***Enabled*** kann ein Objekt aktiviert oder deaktiviert werden. Wenn ***Enabled*** auf **False** gesetzt wird, ist das Element deaktiviert, d.h. es erscheint grau im Dialogfenster und kann nicht mit der Maus angeklickt werden. Wenn ***Enabled*** **True** gesetzt ist, erscheint die übliche Anzeige.

Testen Sie das Dialogfenster in der Entwurfsphase über die Taste F5

4. Initialisieren von Steuerelementen

Durch Einfügen eines Codes in dem **Initialize**-Ereignis können Sie die Steuerelemente initialisieren, bevor Sie das Formular anzeigen. Sie können hier auch die Eigenschaften für die Steuerelemente festlegen, statt das Eigenschaftenfenster zu verwenden. Damit haben Sie die Eigenschaften im Programmtext dokumentiert.

5. Schreiben von Ereignisprozeduren

Alle Steuerelemente besitzen einen vordefinierten Satz von Ereignissen. Beispielsweise besitzt eine Befehlsschaltfläche ein **Click**-Ereignis, dieses tritt auf , wenn der Benutzer auf diese Schaltfläche klickt bzw. ein Textfeld besitzt das **Change**-Ereignis, dieses tritt ein, wenn der Benutzer eine Eingabe macht. Die Ereignisprozeduren werden ausgeführt, wenn die Ereignisse eintreten.

6. Anzeige des Dialogfeldes

Mit der **Show-**Methode kann eine UserForm angezeigt werden.

Überblick über die vorhandenen Steuerelemente

Steuerelement *Beschreibung*	Vorschläge für ***nnn*** bei ***nnnName***	Standard-eigenschaft	Standard-ereignis
Bezeichnungsfeld – Label *Anzeige von Text für Beschriftungen*	***lab***	Caption	Click
Textfeld – TextBox *Eingabe von Texten oder Zahlen (ev. auch Ausgabe)*	***txt***	Value	Change
Rahmen – Frame *Dekoration ausser bei Optionsfeldern*	***frm***	Caption	Click
Befehlsschaltfläche – CommandButton *Ausführen von Prozeduren*	***cmd***	Value	Click
Kontrollkästchen – CheckBox *Ein/Aus-Schalter*	***chk***	Value	Click
Umschaltfeld – ToggleButton *Ein/Aus_Schalter wird angeklickt wie Befehlsschaltfläche, bleibt aber gesetzt wie ein Kontrollkästchen*		Value	Click
Optionsfeld – OptionButton *Eine Option aus einer Gruppe auswählen*	***opt***	Value	Click
Referenzfeld – RefEdit *Eingabe von Bereichen durch Angabe von Zellenbezügen in Tabellenblättern*	***ref***	Value	Change
Listenfeld – ListBox *Auswahl von Einträgen aus einer Liste*	***lst***	Value	Click
Kombinationsfeld – ComboBox *Auswahl von Einträgen aus einer Liste und Eingabe*	***com***	Value (Text)	Click
Bildlaufleiste – ScrollBar *Verschieben der Ansicht in Fenstern und indirekte Eingabe von Werten*	***scr***	Value	Change
Drehfeld – SpinButton *Ändern eines Wertes*	***spb***	Value	Change
Multiseiten – MultiPage *In Fenster blättern*	***mup***	Value	Change
Register – TabStrip *Verschiedene Ansichten erlauben*		Value	Change
Anzeige – Image *Anzeige von Grafik*		Picture	Click

Verwenden von fertigen Beispielen aus der Hilfe

1. Öffnen Sie eine neue EXCEL-Mappe
   → VBE → **EINFÜGEN** → **USERFORM**
   Fügen Sie eine Textbox ein und aktivieren Sie das Eigenschaftenfenster

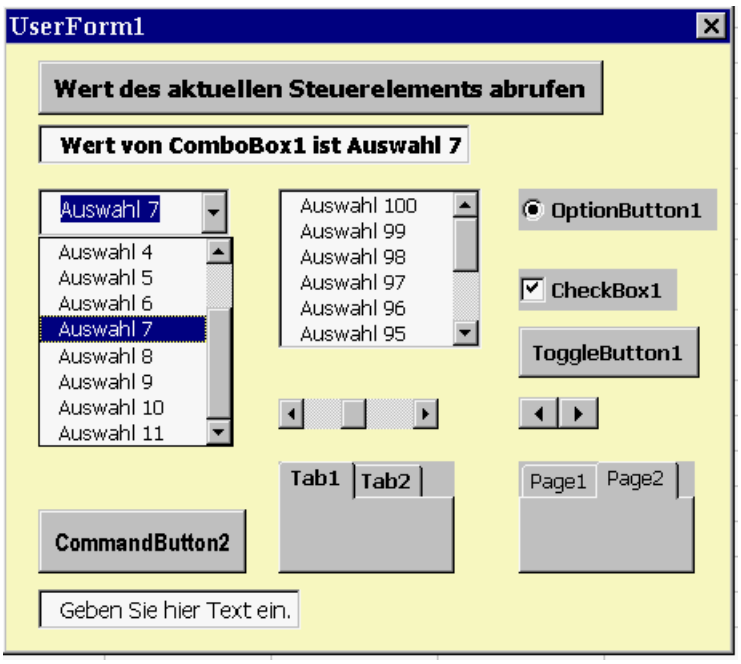
2. Klicken Sie auf die Eigenschaft ***Value*** und Funktionstaste **F1**. Es erscheint der Hilfetext zur Eigenschaft ***Value***.

   Gehen Sie auf das Wort **`Beispiel`** und klicken Sie es an. Sie erhalten eine Seite mit der Überschrift: **Value-Eigenschaft (Beispiel)**. Lesen Sie genau durch, was dieses Beispiel bewirkt.

3. Ergänzen Sie Ihre UserForm um die benötigten Steuerelemente wie das im Hilfebeispiel angegeben ist.

   Kopieren Sie den gesamten Programmcode des Beispiels in das Codefenster:
   Bei **FORMULARE** die ***UserForm1*** aktivieren → **CODE ANZEIGEN** und einfügen.

4. Führen Sie Ihr Makro aus. Wählen Sie zuerst irgendein Steuerelement aus. Durch Klicken auf die entsprechend bezeichnete Befehlsschaltfläche – **CommandButton** können Sie sich den Wert des ausgewählten Steuerelements anzeigen lassen. Probieren Sie das für alle vorhandenen Steuerelemente aus.



5. Falls Sie Interesse an Gestaltungsmöglichkeiten für Rahmen und Farben von einer Text-Box haben, sollten Sie auf die Eigenschaft ***BackColor*** gehen, mit F1 die Hilfe aktivieren und dort auf Beispiel.

In der Hilfe können Sie übrigens eine Unmenge von Beispielen finden!

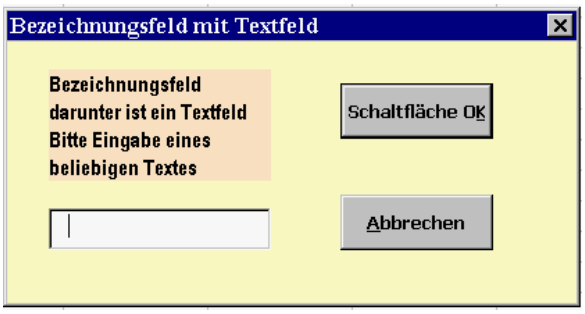
Textfeld – Steuerelement (TextBox) Beispiel: *Textfelder.XLS*

Das Textfeld ist eines der wichtigsten und am häufigsten verwendeten Steuerelemente. Es wird meistens für Eingaben des Anwenders verwendet.

1. Beispiel: Text eingeben und in Tabelle1 übernehmen

Eingabe eines beliebigen **Textes** mittels Dialogbox. Bei Drücken auf **OK** wird die eingegebene Information in *Tabelle1* in die Zelle **A1** geschrieben, die nächste Eingabe kommt in Zelle **A2** etc.

Dialogbox mit **Bezeichnungsfeld** und darunter **Textfeld** und den **zwei Schaltflächen** für OK und Abbrechen erstellen:



Bezeichnungsfeld

Label1 wird zur Anzeige des Textes für das darunterstehende Textfeld verwendet.
Eigenschaften: bei *Caption* wird der gewünschte Text, der im Bezeichnungsfeld stehen soll, eingegeben.

TextBox1 benötigt keine Eigenschaften und keine Prozedur.

CommandButton		**1**	**2**
Eigenschaften:	(Name)	cmdAbbrechen	cmdOK
	Caption	Abbrechen	Schaltfläche OK
	Accelerator	A	K
	Default	False	True
	Cancel	True	False

Erste Schaltfläche – Abbrechen:

Die Schaltfläche mit *Cancel* Eigenschaft ***True***, besitzt die gleiche Wirkung wie die ESC-Taste. Diese Eigenschaft kann nur für eine Schaltfläche definiert werden. Üblicherweise wird diese Eigenschaft immer der Abbrechen-Taste zugewiesen.

Damit das Fenster nach Klicken auf Abbrechen geschlossen wird, ergänzen wir die Ereignisprozedur `Click`, für diese Schaltfläche durch `Unload Me`:

```
Private Sub cmdAbbrechen_Click()
  Unload Me
End Sub
```

Kommentar: das Schlüsselwort **`Me`** ist eine Referenz auf das Dialogfeld-Objekt. Es ist eine Kurzschreibweise, die anstelle des Namens des Dialogfeldes verwendet wird.
Die **`Unload`** – Anweisung lagert ein Objekt aus dem Speicher aus und gibt den belegten Speicher wieder frei.

Zweite Schaltfläche – Schaltfläche OK:

Die *Default* Eigenschaft legt fest, welche Schaltfläche als Standard in einem Fenster definiert ist. Eine solche Schaltfläche wird dann durch das Drücken der Enter-Taste aktiviert, vorausgesetzt, keine weitere Befehlsschaltfläche im Fenster besitzt in diesem Moment den Fokus. Diese Eigenschaft kann nur für eine Schaltfläche definiert werden. Üblicherweise weist man diese Eigenschaft immer der OK-Taste zu.

Durch Drücken der OK-Schaltfläche soll der eingegebene Wert in die Tabelle übertragen werden, daher müssen wir folgende Prozedur beim Click-Ereignis dieser Schaltfläche angeben:

```
 1  Private Sub cmdOK_Click()
 2    Static zeilennr
 3    zeilennr = zeilennr + 1
 4    Worksheets("Tabelle1").Activate
 5    zelle = "A" & zeilennr
 6    Range(zelle).Select
 7    ActiveCell.FormulaR1C1 = TextBox1.Value
 8    MsgBox "Übertrag in Tabelle OK - nächste Eingabe oder Abbrechen"
 9    TextBox1.Value = ""
10    TextBox1.SetFocus
11  End Sub
```

Um das Programm besser kommentieren zu können, wurden die Zeilen numeriert:

Zeile 2: Die Variable `zeilennr` wird für das Schreiben der Werte in die Tabelle benötigt. Die Werte sollen in eine Spalte A1, A2, A3, ... geschrieben werden und `zeilennr` stellt die Zeilenerhöhung dar.

`zeilennr` wird mit dem Wert 0 initialisiert. **`Static`** ist wichtig, sonst wird `zeilennr` jedesmal, wenn man auf OK drückt auf den Anfangswert zurückgesetzt und es soll aber automatisch um 1 erhöht werden, um die eingegebenen Werte in die aufeinanderfolgenden Zeilen der Tabellenspalte A zu bringen.

Zeile 3 Zeilenzähler um 1 erhöhen

Zeile 4: Aktivieren des angegebenen Tabellenblatts

Zeile 5: Koordinaten der Zelle in der Tabelle für den eingegebenen Wert zusammenbasteln. Bestimmung der Spalte A und der entsprechenden Zeile, das ist die A1-Schreibweise oder das A1-Bezugssystem.

Zeile 6: über das Objekt **`Range`** und die Methode **`Select`** wird die entsprechende Zelle aktiviert d.h. der Cursor wird in dieser Zelle positioniert.

Zeile 7: **`ActiveCell.FormulaR1C1`** bedeutet, die aktive Zelle bekommt über die Eigenschaft FormulaR1C1 einen Inhalt, das wirkt wie die manuelle Eingabe in die Zelle. Hier erhält die Zelle, den in das Textfeld (TextBox1) eingegebenen Wert (`Value`).

Zeile 9-10: Inhalt vom Textfeld löschen und Focus hinbringen.

Beim Aktivieren der Dialogbox wird Tabelle 1 aktiviert und gelöscht:

```
Private Sub UserForm_Initialize()
  Worksheets("Tabelle1").Activate
  Worksheets("Tabelle1").Cells.Select
  Selection.Clear
  Range("A1").Select
End Sub
```

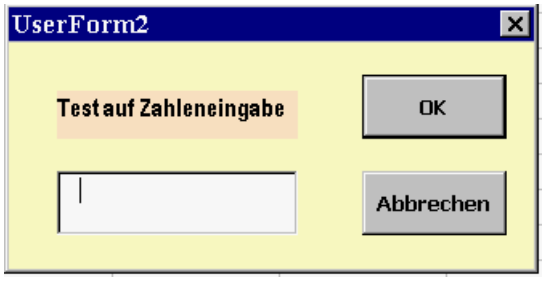
2.Beispiel: Zahl eingeben und in Tabelle2 übernehmen

In Ergänzung zu dem vorigen Beispiel wird hier abgefragt, ob überhaupt etwas eingegeben wurde. Es wird anschliessend mit Hilfe der Funktion **IsNumeric** geprüft, ob eine Zahl eingegeben wurde. Die Zahl wird in das numerische Format double umgewandelt bevor sie in Tabelle2 übertragen wird.

Testen Sie dieses Beispiel mit positiven und negativen Zahlen und auch mit Dezimalzahlen z.B. 1.234 oder 1.2 E-20 etc.

Die Prozedur nach dem Klicken auf OK wird etwas abgeändert:

Falls keine / oder eine falsche Eingabe in das Textfeld gemacht wurde und OK geklickt wurde, erscheint eine Fehlermeldung, das Textfeld wird geleert, der Focus geht zurück in das Textfeld für eine neue Eingabe und die Prozedur wird mit **`Exit Sub`** beendet. Alle anderen Änderungen sind in der Liste kommentiert.

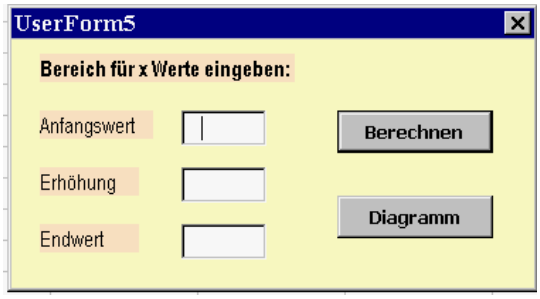



```
Private Sub cmdOK_Click()
  Static zeilennr As Byte
  Dim eingabezahl As Double
  Dim zelle As String
  If TextBox1.Value = "" Then
    ' der Anwender hat nichts eingegeben und OK geklickt
    MsgBox "Bitte eine Zahl eingeben oder Abbrechen klicken!"
    TextBox1.Value = ""
    TextBox1.SetFocus
    Exit Sub
  End If
  If IsNumeric(TextBox1.Value) Then
    eingabezahl = CDbl(TextBox1.Value)
    ' die Zahl wird in das grösste numerische Format umgewandelt
  Else
    MsgBox "Nur Zahlen eingeben"
    TextBox1.Value = ""
    TextBox1.SetFocus
    Exit Sub
  End If
  zeilennr = zeilennr + 1
  Worksheets("Tabelle2").Activate
  zelle = "B" & CStr(zeilennr)
  Range(zelle).Select
  ActiveCell.FormulaR1C1 = eingabezahl
  TextBox1.Value = ""
  TextBox1.SetFocus
End Sub
```

3.Beispiel *Wertetabelle in Tabelle3 erzeugen*

Für die Erstellung einer Wertetabelle für eine Funktion werden über eine Dialogbox Anfangswert, Erhöhung und Endwert für die Variable **x** eingegeben, in Tabelle3 wird die Wertetabelle erzeugt. Zur Illustration ist das Programm für die Funktion **y = sin(x)** angegeben. Die Erstellung von Überschriften in der Tabelle, sowie das Zeichnen des Diagramms sind nicht programmiert.

Wir benötigen hier drei Textfelder für Anfangswert (xa), Erhöhung (dx) und Endwert (xe) und die Schaltfläche Berechnen.




```
Dim xa As Variant, dx As Variant, xe As Variant
  ' globale Variable

Private Sub txtdx_Change()
  dx = txtdx.Value
End Sub

Private Sub txtxe_Change()
  xe = txtxe.Value
End Sub

Private Sub txtxa_Change()
  xa = txtxa.Value
End Sub

Private Sub cmdBerechnen_Click()
  Dim x As Double, y As Double, zähler As Integer
  Range("a1").Select
  ' Überschriften erstellen und formatieren
  zähler = 2
  x = xa
  Do
    zähler = zähler + 1
    Worksheets("Tabelle3").Cells(zähler, 1).Value = Format(x, "0.00")
    ' die folgende Zeile zeigt die Verwendung der Funktion sin(x)
    Worksheets("Tabelle3").Cells(zähler, 2).Value = "=SIN(RC[-1])"
    x = x + dx
  Loop Until x > xe + 0.0001
  Worksheets("Tabelle3").Activate
  Unload Me
End Sub
```

Die Übertragung der x- Werte und der y-Werte in die Tabelle erfolgt mit Hilfe einer DO .. Loop Until, wobei die Zellen mit Hilfe der **Cell – Eigenschaft** angesprochen werden. Der **erste Index** gibt die **Zeilennummer**, der **zweite** die **Spalte** an. Wir verwenden Spalte A für die x-Werte und Spalte B für die y-Werte und beginnen die Wertetabelle in der 3. Zeile.

4. Beispiel: Inventarliste in Tabelle4 erzeugen

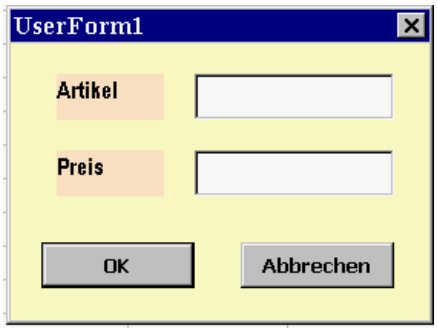
Dieses Beispiel zeigt, wie man die Eingabe in eine EXCEL-Tabele über eine Dialogbox vornehmen kann. Wir wollen nur die Angaben für Artikelbezeichnung und Preis in die Inventarliste übernehmen. Weitere Felder können in ähnlicher Weise eingefügt werden. Die Gestaltung der Tabelle und Berechnungen in der Tabelle sind nicht programmiert.

Die Übernahme der eingegebenen Werte erfolgt hier über **relative Adressierung.** Sie können auch selbst Makros mit relativem Bezug aufzeichnen lassen. Dazu müssen Sie nach dem Start der Aufzeichnung die Symbolleiste **AUFZEICHNUNG BEENDEN** aktivieren, dort finden Sie die Schaltfläche **RELATIVER BEZUG**.

Die ***Offset***-Eigenschaft dient dazu, auf eine Zelle zu verweisen, die in relativem Bezug zu einer anderen Zelle steht. z.B. `Offset(1,3)` bedeutet einen Verweis auf eine Zelle, die sich eine Zeile unter und drei Spalten rechts der aktiven Zelle befindet.

Die Prozedur InitDialog leert die Eingabefelder und setzt den Cursor auf die Artikeleingabe:

```
Sub InitDialog()
  txtArtikel.Value = ""
  txtPreis.Value = ""
  txtArtikel.SetFocus
End Sub
```



Bei der Initialisierung der UserForm wird ein neues Tabellenblatt mit dem Namen Tabelle4 eingefügt:

```
Private Sub UserForm_Initialize()
  Sheets("Tabelle4").Select
  Sheets("Tabelle4").Name = "Tabelle4"
  Worksheets("Tabelle4").Activate
  Worksheets("Tabelle4").Cells.Select
  Selection.Clear
  Range("A1").Select
  tabellenkopf
End Sub
```

In dieser Prozedur wird die Prozedur tabellenkopf verwendet, die Sie auch über Makroaufzeichnung erhalten können:

```
Sub tabellenkopf()
 Range("a1").Select
 ActiveCell.FormulaR1C1 = "Inventarliste"
 Range("a2").Select
 ActiveCell.FormulaR1C1 = "Artikel"
 Range("b2").Select
 ActiveCell.FormulaR1C1 = "Preis"
 Range("a3").Select
End Sub
```

Beim Drücken auf die Ok-Schaltfläche wird folgende Prozedur ausgeführt:

```
Private Sub cmdOk_Click()
  ActiveCell.FormulaR1C1 = txtArtikel.Value
  ActiveCell.Offset(0, 1).Activate       'eine Spalte nach rechts
  ActiveCell.FormulaR1C1 = txtPreis.Value
  ActiveCell.Offset(1, -1).Activate
  'eine Zeile hinunter und eine Spalte nach links
  InitDialog
End Sub
```

Referenzfeld – Steuerelement (RefEdit) Beispiel: *Referenzleld.XLS*

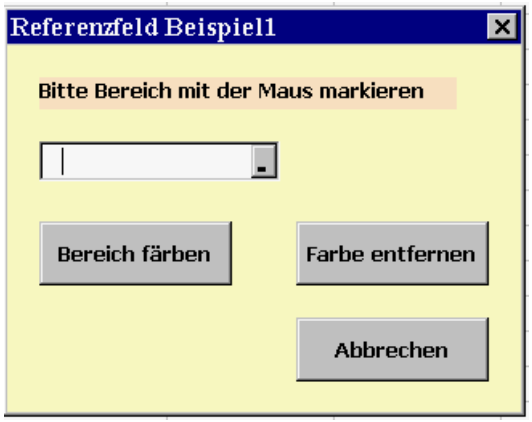
Das Referenzfeld ist eine der besten Neuheiten von EXCEL ab Office 97, da seine Einführung dem Programmierer eine Menge an Programmcode erspart. Dieses Steuerelement wurde speziell für Excel eingeführt und steht auch nur dort zur Verfügung, nicht aber in den anderen Anwendungen des Programmpakets Office 97.

Mit dem Referenzfeld kann man in einfacher Weise einen Bereich einer Tabelle mittels Markierung mit der Maus übergeben. In dieser Art werden auch bei Funktionsaufrufen die Bereiche eingegeben. Es wird in sehr vielen Demo-Beispielen verwendet.

Die Handhabung des Referenzfeldes in einem Programm ist genau die gleiche wie die für das Textfeld. Die zwei Elemente besitzen die gleichen Standardeigenschaften und Standardereignisse. Das Referenzfeld gibt einen Bereich als Variant / String zurück

1. Beispiel: Beliebigen Bereich einer Tabelle färben / Farbe entfernen

Wir verwenden ein Referenzfeld mit Beschriftung und 3 Schaltflächen. Da der markierte Bereich wahlweise gefärbt oder ohne Farbe erscheinen soll, benötigen wir die globale Variable `bereich`. Sobald eine der Schaltflächen gedrückt wird, wird das Referenzfeld verlassen und über das Ereignis Exit wird in der angegebenen Prozedur der Inhalt vom Referenzfeld in die Variable `bereich` übertragen.




```
Dim bereich As String

Private Sub cmdFarbe_Click()
  Range(bereich).Interior.ColorIndex = Int((56 * Rnd) + 1)
End Sub

Private Sub cmdOhneFarbe_Click()
  Range(bereich).Interior.ColorIndex = xlNone
End Sub

Private Sub refBereich_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  bereich = refBereich.Text
End Sub

Private Sub cmdAbbrechen_Click()
  Unload Me
End Sub
```

2. Beispiel *Referenzfeld mit Aufnahme mehrerer Bereiche*

Bekanntlich kann man in einer Tabelle mehrere nicht zusammenhängende Bereiche mit der Maus unter Verwendung der Taste Ctrl markieren. Wenn man in einem Referenzfeld einen mehrfachen Bereich markiert, werden die einzelnen Bereiche mit dem Semikolon - Zeichen **";"** getrennt. Im VBA-Programm hat aber das Range-Objekt eine andere Syntax: hier ist das Trennzeichen zwischen den einzelnen Teilen das Komma - Zeichen **","** . Es ist daher notwendig, die in der Zeichenkette als Trennzeichen verwendeten Strichpunkte durch Kommata zu ersetzen. Unser Beispiel zeigt eine Funktion **`setBereich`**, welche die Trennzeichen des Referenzfeldes ändert. Diese Funktion können Sie selbstverständlich in Ihre eigenen VBA-Programme übernehmen.

Erklärungen zur Funktion **`setBereich`**:

Die Funktion bekommt als Parameter den Bereich des Referenzfeldes. Der geänderte Bereich selbst wird als Rückgabewert des Typs String der aufrufenden Funktion zurückgegeben.

Die **`For`** Schleife wird für jedes Zeichen der Zeichenkette ausgeführt, die den Bereich darstellt. Die Länge der Zeichenkette wird von der Funktion **`Len`** ermittelt.

Die Funktion **`Mid$`** gibt aus der Zeichenkette ***bereich*** ab der Position **`i`** genau 1 Zeichen zurück. Falls dieses Zeichen ein Semikolon ist, wird der **`Then`**-Zweig ausgeführt. In diesem wird die Zeichenkette in die links und rechts vom Semikolon stehenden Teile geteilt. Der linke Teil wird mit der **`Mid$`**-Funktion geliefert, der rechte Teil mit der **`Right$`**-Funktion. Zwischen die beiden Teile wird ein Komma gesetzt.

Zur Illustration betrachten Sie bitte folgendes Beispiel:

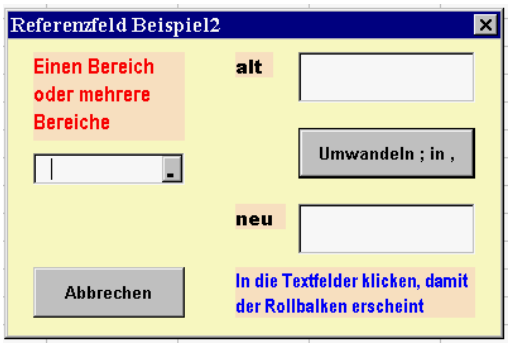
Tabelle1!A2:C6;Tabelle2!F3:K9

Das Semikolon befindet an der Stelle 15, daher umfasst der linke Teil 14 Zeichen, da die Gesamtlänge 29 Zeichen beträgt, hat der auch der rechte Teil eine Länge von 29 – 15 = 14 Zeichen.

Programmliste für diese Funktion:

```
Private Function setBereich(ByVal bereich As String) As String
  Dim i As Byte
  For i = 1 To Len(bereich)
    If Mid$(bereich, i, 1) = ";" Then
      bereich = Mid$(bereich, 1, i - 1) & "," & Right$(bereich, _
                Len(bereich) - i)
    End If
  Next i
  setBereich = bereich
End Function
```

Um diesen Vorgang besser zu illustrieren haben wir folgende Dialogbox erstellt:



Nach dem Erscheinen der Dialogbox klicken wir in das Referenzfeld auf den kleinen Balken, damit kommen wir in die EXCEL Arbeitsmappe, markieren dort mit der Maus einen Bereich, bzw. mehrere Bereiche (Ctrl-Taste). Nach der Anzeige der markierten Bereiche im Referenzfeld wird durch das Change-Ereignis durch die Prozedur **`Sub refBereich_Change`** die Beschriftung des oberhalb stehenden Textfeldes geändert.

```
Private Sub refBereich_Change()
  labRef.Caption = "Bitte mit dem Tabulator weiter!"
End Sub
```

Wenn man die Tab-Taste drückt, wird das Referenzfeld verlassen, über das Exit-Ereignis wird durch die Prozedur **`Sub refBereich_Exit`** der Inhalt vom Referenzfeld in das Textfeld übertragen.

```
Private Sub refBereich_Exit(ByVal Cancel As MSForms.ReturnBoolean)
  bereich = RefBereich.Text
  txtText1.Text = bereich
End Sub
```

Damit man die gesamte Bereichsangabe lesen kann, muss man das Feld anklicken. Dadurch wird ein Rollbalken angezeigt. Im Eigenschaftenfenster wurde für die TextBox1 die Eigenschaft *ScrollBars* auf **True** gesetzt. Sie finden übrigens eine sehr gute Illustration zu den Rollbalken in der Hilfe – ScrollBars-, KeepScrollBarsVisible-Eigenschaften (Beispiel) –, das Sie z.B. über die ScrollBars-Eigenschaft unter Beispiel finden, übernehmen und direkt ausprobieren können.

Beim Klicken auf die Schaltfläche **UMWANDELN** wird die oben erklärte Funktion ausgeführt und der geänderte Text im unterem Textfeld angezeigt.

```
Private Sub cmdUmwandeln_Click()
  bereich = setBereich(bereich)
  'MsgBox Bereich
  txtText2.Text = bereich
End Sub
```

Beachten Sie, dass die Variable **`bereich`** als globale Variable im Deklarationsteil – also am Anfang des Codes – deklariert werden muss: `Dim bereich As String`

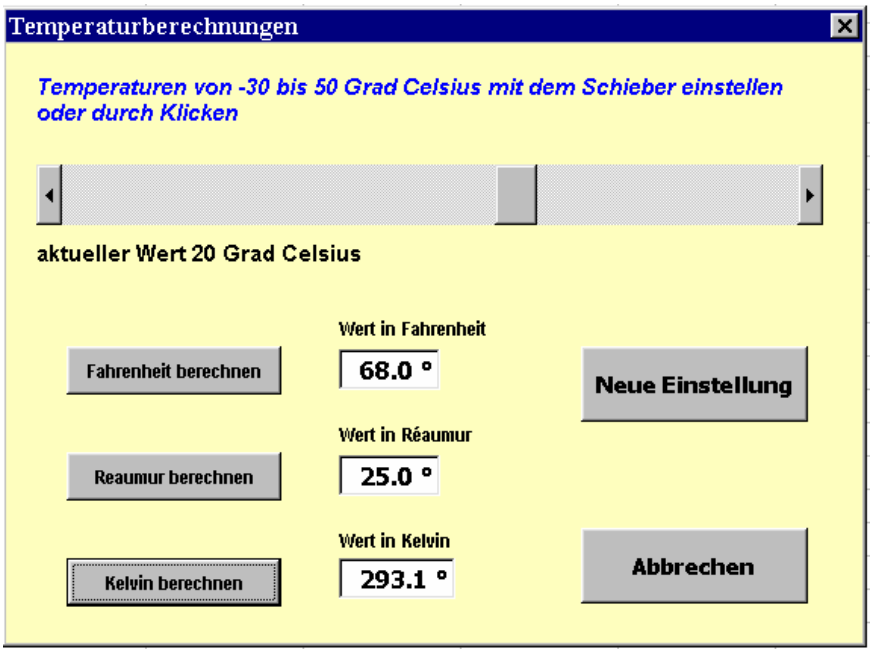
Bildlaufleiste – Steuerelement (ScrollBar) Beispiel: *Bildlaufleiste.XLS*

Eine Bildlaufleiste-Steuerelement ist ein eigenständiges Steuerelement, das man in einem Dialogfeld vertikal oder horizontal positionieren kann. Optisch sieht es genauso aus wie die Bildlaufleiste oder der Rollbalken im Textfeld oder in Listen- / oder Kombinationsfeldern.

Die Leiste ist an einen ganzzahligen (Integer) Bereich gekoppelt, dessen Grenzen man bei den Eigenschaften ***Max*** und ***Min*** angeben kann. Jede Position des Schiebers auf der Leiste entspricht einem bestimmten Wert des festgelegten Intervalls. Der Schieber kann über einen Klick auf die Schaltflächen mit den Pfeilen am oberen und am unteren Ende der Leiste bewegt werden (Eigenschaft ***SmallChange***) oder mit einem Klick direkt auf die Leiste (***LargeChange***) oder mit dem Schieber selbst. Über die ***Value***-Eigenschaft haben wir Zugriff zu dem aktuell eingestellten Wert.

1. Beispiel: *Temperatur in Grad Celsius über eine Bildlaufleiste einstellen, Umrechnungsmöglichkeit in Fahrenheit, Reaumur und Kelvin*

Wir erstellen eine Dialogbox mit einer Bildlaufleiste mit zwei dazugehörigen Bezeichnungsfeldern, drei Schaltflächen für die gewünschten Berechnungen, drei Textfelder für die Anzeige der Umrechnungsergebnisse mit entsprechenden Beschriftungen und die Schaltflächen **NEUE EINSTELLUNG** und **ABBRECHEN**.



Der Bereich für die Einstellung der Celsiusgradwerte wurde auf -30 bis 50 beschränkt mit den beiden Schrittweiten 1 und 5 Grad. Beachten Sie, dass beim Change-Ereignis der aktuelle Wert in die globale Variable Celsius gebracht wird und ausserdem in das Bezeichnungsfeld, das sich unterhalb der Leiste befindet.

Bei den Textfeldern für die Anzeige der Ergebnisse wurde Eigenschaft ***Autosize*** auf **True** gesetzt, das bewirkt automatische Anpassung an die tatsächlich benötigte Grösse. Auch die Eigenschaft ***Locked*** wurde auf **True** gesetzt, dadurch ist es für eine Eingabe gesperrt.

Für eine Neueinstellung werden die Inhalte der Textfelder gelöscht und der Cursor wird in unser "Thermometer" gebracht und blinkt dort kräftig.

Es folgt die gesamte Programmliste wie sie im Code der UserForm zu finden ist:

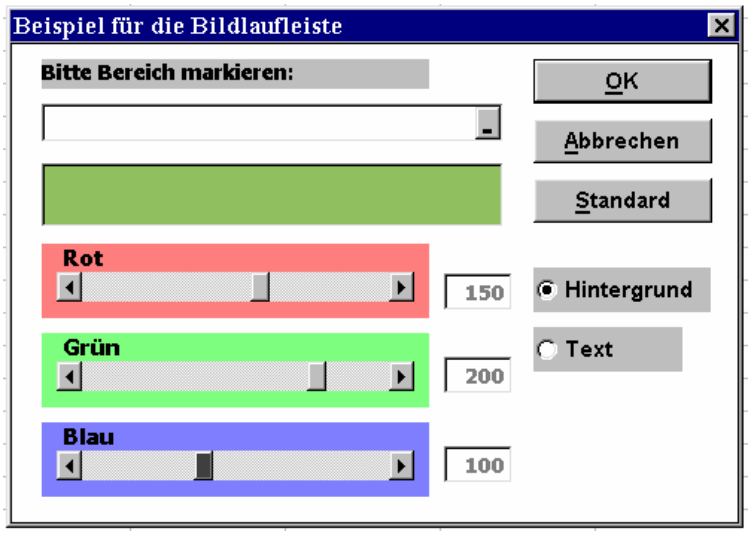
```
Option Explicit

' Deklarationen

Dim Celsius As Integer

Function Fahrenheit(GradCelsius)
  Fahrenheit = 32 + 9 / 5 * GradCelsius
End Function

Function Reaumur(GradCelsius)
  Reaumur = 5 * GradCelsius / 4
End Function

Function Kelvin(GradCelsius)
  Kelvin = GradCelsius - 273.15
End Function

' Ende der Deklarationen

Private Sub scbCelsius_Change()
  Celsius = scbCelsius.Value
  lblCelsiusAnzeige.Caption = "aktueller Wert " & Celsius & " Grad Celsius"
End Sub

Private Sub cmdFahrenheit_Click()
  txtFahrenheit = Format(Fahrenheit(Celsius), "0.0 °")
End Sub

Private Sub cmdReaumur_Click()
  txtReaumur = Format(Reaumur(Celsius), "0.0 °")
End Sub

Private Sub cmdKelvin_Click()
  txtKelvin = Format(Kelvin(Celsius), "0.0 °")
End Sub

Private Sub cmdNeueinstellen_Click()
  txtReaumur = ""
  txtFahrenheit = ""
  txtKelvin = ""
  scbCelsius.SetFocus
End Sub

Private Sub cmdAbbrechen_Click()
  Unload Me
End Sub
```

2.Beispiel: Bereiche farblich markieren, Farbe in einem Dialogfenster mischen

Ein zusammenhängender Tabellenbereich oder mehrere aus getrennten Teilen bestehende Tabellenbereiche, die sich auch in verschiedenen Tabellen befinden können, werden mit einer eigenen Farbmischung gefärbt, die aus den drei Grundfarben rot, grün und blau mit Hilfe der Bildlaufleisten hergestellt wird. Damit lassen sich andere Farben als die Palettenfarben herstellen.

Übersicht über die verwendeten Steuerelemente:

- **Referenzfeld** für die Eingabe des Tabellenbereichs mit darüberstehendem Bezeichnungsfeld; es können auch nicht zusammenhängende Bereiche eingegeben werden.
- **Textfeld** zur Anzeige der Farbmischung, dieses Textfeld wird über die Eigenschaft ***Locked*** → **True** für die Eingabe gesperrt.
- Drei farbige **Rahmen (**Frame**)** mit **Bildlaufleisten**:
  Über die Eigenschaft ***KeepScrollBarsVisible*** mit **3-frmSrollBarsBoth** werden innerhalb dieser Rahmen die Bildlaufleisten angezeigt.
  Über die Eigenschaften ***Min(0), Max(255), LargeChange(15)*** und ***SmallChange(5)*** werden die Werte der Bildlaufleisten gesetzt.
- Drei **Textfelder** zur Anzeige der aktuellen gewählten Farbnummern.
- Zwei **Optionsfelder** zur Auswahl, ob **TEXT** oder **HINTERGRUND** gefärbt werden soll.
- Drei **Schaltflächen**:
  OK zur Übernahme der ausgewählten Farbmischung, das Fenster bleibt offen, damit noch weitere Bereiche gefärbt werden können. **STANDARD**: Text schwarz und keine Farbe für Hintergrund und **ABBRECHEN**.



Beschreibung der Prozeduren:

Auf Modulebene erfolgt die Deklaration der drei globalen Farbvariablen. Dann erfolgt die Deklaration der **Funktion `setBereich`**, die nur für den Fall der Eingabe von nicht zusammenhängenden Tabellenbereichen benötigt wird, um das über das Referenzfeld eingegebene Trennzeichen "**;**" zwischen den Bereichen in ein "**,**" umzuwandeln, wie es von VBA für die Range-Syntax benötigt wird.

Die Sub-Prozedur **`setAnzeige`** zeigt die aktuelle Farbmischung im Textfeld ***AnzeigeBox*** an. Die Hintergrundfarbe des Textfeldes wird mit der Funktion RGB geändert. Ein RGB-Farbwert gibt die relative Intensität von Rot, Grün und Blau an, um eine bestimmte Farbe hervorzurufen. Ausserdem werden die Zahlenwerte der drei Bildlaufleisten in den entsprechenden drei kleinen Textfeldern angezeigt. Bei den Eigenschaften wurde bei diesen Textfeldern ***Enabled*** auf ***False*** gesetzt, daher erscheinen diese Zahlen grau und können nicht mit der Maus angeklickt werden, die Objekte werden dadurch deaktiviert.

Prozedur **`Initialize`** der **UserForm**:
Alle Steuerelemente erhalten ihre Startwerte; die Schieber der Bildlaufleisten und die Farbvariablen werden auf 0 gesetzt und die Farbe schwarz wird angezeigt, der Cursor blinkt im Referenzfeld und die Option **Hintergrund** wird aktiviert.
Die Eigenschaft ***BackColor*** wird auf *Fensterhintergrund keine Farbe* gesetzt, damit die Anzeige der Farbe möglichst gut mit der Farbe auf dem Tabellenblatt übereinstimmt.

Ereignisprozeduren **`Change`** für die **Bildlaufleisten**:
Bei jeder Bewegung des Schiebers erfolgt eine Änderung der Werte zwischen 0 und 255. Der aktuelle Wert der Bildlaufleiste ist in der Eigenschaft ***Value*** gespeichert. Dieser Wert wird der globalen Farbvariablen zugewiesen.

Prozedur **`Click`** für Schaltfläche **OK**:
Zuerst wird geprüft, ob der Anwender eine Eingabe gemacht hat. Falls keine Eingabe gemacht wurde, erscheint eine Meldung und der Cursor wird zurück in das Referenzfeld geschickt. In diesem Fall wird die Prozedur vorzeitig mit **`Exit`** beendet. Durch die Funktion **`setBereich`** wird der eingegebene Tabellenbereich der Stringvariablen zugewiesen, wobei eventuell vorhandene Trennzeichen ";" durch "," ersetzt werden. Anschliessend werden in Abhängigkeit der gesetzten Option **Hintergrundfarbe** oder **Textfarbe** auf die aktuellen Werte der globalen Farbvariablen gesetzt. Der Cursor wird dann ins Eingabefeld gebracht, um eine weitere Eingabe zu erlauben.

Prozedur **`Click`** für Schaltfläche **STANDARD**:
Diese Prozedur ist ähnlich der Prozedur für OK, nur werden hier die Farben auf Standard gesetzt, d.h. Hintergrund auf keine Farbe (x1ColorIndexNone) und Textfarbe auf schwarz. Damit lassen sich gewisse Spielereien wieder rückgängig machen.

Programmliste:

```
' Deklarationen auf Modulebene
Option Explicit
Private rot As Integer
Private gruen As Integer
Private blau As Integer

Private Function setBereich(ByVal Bereich As String) As String
  Dim i As Integer
  For i = 1 To Len(Bereich)
    If Mid$(Bereich, i, 1) = ";" Then
      Bereich = Mid$(Bereich, 1, i - 1) & "," & Right$(Bereich, _
                Len(Bereich) - i)
    End If
  Next i
  setBereich = Bereich
End Function
```

```
Sub setAnzeige()
  Anzeigebox.BackColor = RGB(rot, gruen, blau)
  Rotbox.Value = rot
  Gruenbox.Value = gruen
  Blaubox.Value = blau
End Sub

Private Sub UserForm_Initialize()
  RotScrollBar.Value = 0
  GruenScrollBar.Value = 0
  BlauScrollBar = 0
  rot = 0
  blau = 0
  gruen = 0
  setAnzeige
  RefEdit1.SetFocus
  OptHinterg.Value = True
End Sub

Private Sub RotScrollBar_Change()
  rot = RotScrollBar.Value
  setAnzeige
End Sub

Private Sub GruenScrollBar_Change()
  gruen = GruenScrollBar.Value
  setAnzeige
End Sub

Private Sub BlauScrollBar_Change()
  blau = BlauScrollBar.Value
  setAnzeige
End Sub

Private Sub OKButton_Click()
  Dim Bereich As String
  Dim i As Integer
  If RefEdit1.Value = "" Then
    MsgBox "Bitte Bereich markieren!"
    RefEdit1.SetFocus
    Exit Sub
  End If
  Bereich = setBereich(RefEdit1.Value)
  If OptHinterg.Value Then
    Range(Bereich).Interior.Color = RGB(rot, gruen, blau)
  Else
    Range(Bereich).Font.Color = RGB(rot, gruen, blau)
  End If
  RefEdit1.SetFocus
End Sub
```

```
Private Sub DefaultButton_Click()
  Dim Bereich As String
  If RefEdit1.Value = "" Then
    MsgBox "Bitte Bereich markieren!"
    RefEdit1.SetFocus
    Exit Sub
  End If
  Bereich = setBereich(RefEdit1.Value)
  If OptHinterg.Value Then
    Range(Bereich).Interior.ColorIndex = xlColorIndexNone
  Else
    Range(Bereich).Font.ColorIndex = xlColorIndexAutomatic
  End If
  RefEdit1.SetFocus
End Sub

Private Sub CancelButton_Click()
  Unload Me
End Sub
```

Befehlsschaltflächen – Steuerelement (CommandButton) *Befehlsschaltfläche.XLS*

Befehlsschaltflächen sind in jedem Dialogfeld vorhanden. Im allgemeinen verwendet man mindestens die Schaltflächen **OK** und **ABBRECHEN**. Es können aber beliebig viele andere Schaltflächen hinzugefügt werden.

Eigenschaften der Schaltflächen:

CommandButton	**OK**	**ABBRECHEN**
(Name)	**cmdOk**	**cmdAbbrechen**
Caption	OK	Abbrechen
Accelerator	K	A
Default	True	False
Cancel	False	True

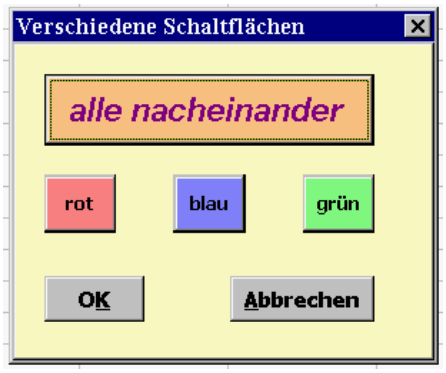
Im allgemeinen benötigt man für die Schaltfläche Abbrechen folgende Prozedur:

```
Private Sub cmdAbbrechen_Click()
  Unload Me
End Sub
```

1. Beispiel: Eine Schaltfläche aktiviert andere Schaltflächen

Bei diesem Beispiel werden verschiedene Farbeigenschaften (Hintergrundfarbe und auch Textfarbe) verwendet. Die drei kleinen Schaltflächen können einzeln aktiviert werde. Durch Anklicken der grossen Schaltfläche werden sie alle drei nacheinander aktiviert.

Der Wert einer Schaltfläche kann durch Klicken oder auch in einer Prozedur durch Zuweisung auf **True** gesetzt werden. Ansonsten sind Schaltflächen immer auf den Wert **False** gesetzt. Das kann in komplexen Programmen verwendet werden: statt eine Menge von Schaltfläche anzuklicken, könnte man dann eine einzige verwenden, die alle anderen aktiviert.



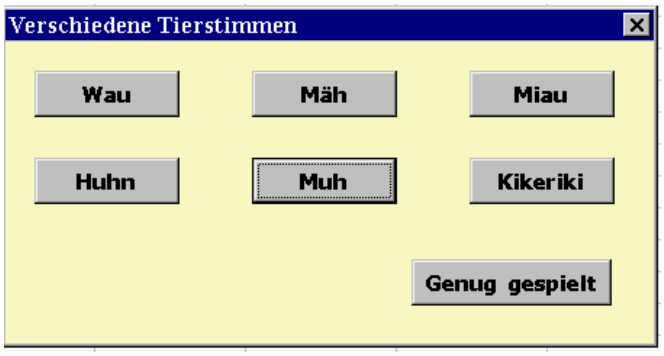
Programmliste:

```
Private Sub cmdAlles_Click()
  cmdRot.Value = True
  cmdBlau.Value = True
  cmdGruen.Value = True
End Sub

Private Sub cmdBlau_Click()
  MsgBox "Ich bin blau"
End Sub
```

```
:
:

Private Sub cmdOK_Click()
  MsgBox "Ende"
End Sub
```

2. Beispiel *Tierstimmen erzeugen*

Dieses Beispiel können Sie nur ausführen, wenn Sie Sound und die nötigen Wave Dateien auf Ihrem PC haben.



In einem **Modul** befindet sich die notwendige Deklaration und die Definition der Funktion **`PlaySound`**, in der auch getestet wird, ob die gewünschten Dateien vorhanden sind.

```
Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
(ByVal WAVDatei As String, ByVal Wiedergabemodus As Long) As Long

Public Function Playsound(SoundDatei)
  If Dir$(SoundDatei) > "" Then
    sndPlaySound SoundDatei, 1
    Playsound = SoundDatei
  Else
    Playsound = "Datei nicht gefunden"
  End If
End Function
```

In der **UserForm** sind die einzelnen Click-Prozeduren:

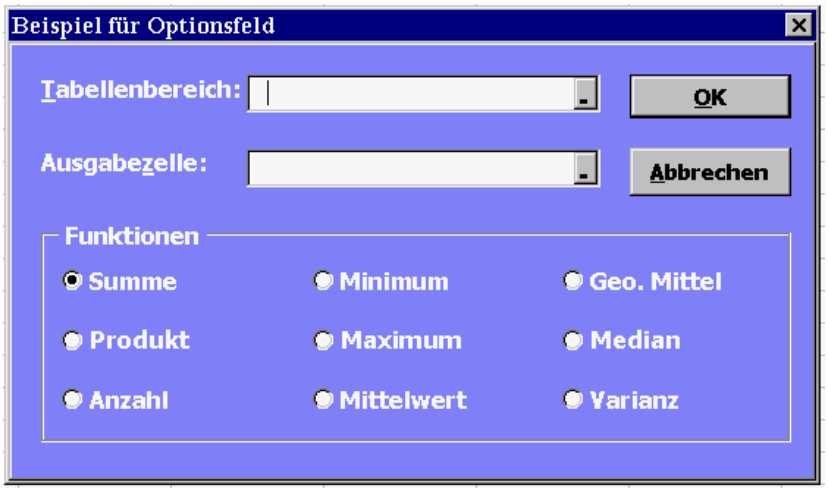
```
Private Sub cmdHund_Click()
  Playsound "C:\winnt\media\hund.wav"
End Sub

Private Sub cmdHuhn_Click()
:
:
:
Private Sub cmdSchaf_Click()
  Playsound "C:\winnt\media\schaf.wav"
End Sub

Private Sub cmdOK_Click()
  MsgBox "Danke für die Vorstellung"
  Unload Me
End Sub
```

Optionsfeld – Steuerelement (OptionButton) Beispiel: *Optionsfelder.XLS*

Ein Optionsfeld zeigt an, ob ein Element in einer Gruppe von Auswahlmöglichkeiten ausgewählt wurde oder nicht. Optionsfelder sind eng verwandt mit den Kontrollkästchen oder den Umschaltfelder. Zum Unterschied von diesen können Optionsfelder aber nur zwei Zustände annehmen. Durch einen Klick kann ein Optionsfeld aktiviert werden, in dem runden Feld erscheint dann ein schwarzer Punkt, dies entspricht dem Zustand **True**. Wenn das Feld des Optionsfeldes leer und weiss ist, so entspricht das dem Zustand **False**.

Beispiel: *Aufruf verschiedener Funktionen*

Für einen beliebigen Bereich einer Tabelle, der Zahlen enthält, werden nacheinander verschiedene Funktionen berechnet. Es ist das Ziel, sich für einen gleichbleibenden Bereich, den wiederholten Aufruf des Funktions-Assistenten zu ersparen. Es wird nur jedesmal eine neue Ausgabezelle in der Tabelle gewählt.



Übersicht über die verwendeten Steuerelemente:

Zwei **Referenzfelder** mit dazugehörigen Bezeichnungsfeldern für die Angabe des Tabellenbereichs und der Ausgabezelle

Rahmen mit Bezeichnung Funktionen, der **9 Optionsfelder** enthält, von denen jeweils eine Option ausgewählt werden kann.

Zwei **Schaltflächen:** OK zur Übertragung des Funktionswertes in die Ausgabezelle in der Tabelle und Schaltfläche ABBRECHEN zum Abbrechen des Dialogfeldes.

Beschreibung der Prozeduren:

Auf Modulebene erfolgt die Deklaration der globalen Variablen **`OptNr`** . Beim Klick auf eine Option wird dieser Variablen ein bestimmter Wert zugeordnet.

Prozedur **`Initialize`** der UserForm

Damit die Funktion Summe als Standardfunktion beim Starten des Dialogfeldes markiert ist, muss die Option für die Funktion Summe aktiviert werden. Das erfolgt durch Setzen der Eigenschaft ***Value*** auf **True**. Der Wert der globalen Variablen **`OptNr`** wird auf 1 gesetzt.

Ereignisprozedur **Change** für die Ausgabezelle

Nachdem eine Ausgabezelle gewählt worden ist, soll der Cursor in diese Zelle gebracht werden. Durch die Methode **Select** wird diese Zelle markiert und zur Aufnahme des berechneten Funktionswertes vorbereitet.

Ereignisprozeduren **Click** für alle Optionsfelder

Bei jedem Klick auf eine Option erhält die globale Variable **OptNr** einen bestimmten Wert.

Prozedur **Click** für die Schaltfläche **OK**

Zu Beginn wird geprüft, ob der Anwender den Tabellenbereich und den Ausgabebereich angegeben hat. Falls einer der beiden Bereiche fehlt, erscheint eine Fehlermeldung, der Cursor wird in das benötigte Referenzfeld geschickt und die Subroutine wird vorzeitig mit **Exit** beendet. Wenn die Ausgabezelle nicht leer ist, wird sie markiert. Der vom Benutzer mit der Maus markierte Tabellenbereich ist in der Eigenschaft ***Value*** von diesem Referenzfeld zu finden und wird der lokalen Stringvariablen **bereich** zugewiesen. Mit Hilfe einer Select-Anweisung wird über die Variable **OptNr** die gewünschte Funktion ausgewählt. Es wird die ***Formula*** Eigenschaft verwendet, um in eine Zelle eine richtige Formel einzutragen wie z.B. **SUM(B1:B6)**. Die Wirkung ist identisch mit der Anwendung des Funktionsassistenten. Der Ausdruck für den Funktionsaufruf muss als Text übergeben werden. VBA verlangt den englischen Funktionsnamen, die Variable **bereich** enthält den vom Anwender markierten Bereich.
Das Ergebnis wird in die aktive Zelle gebracht und die Ausgabezelle im Dialogblatt wird geleert und bekommt den Focus.

Beachten Sie, dass Sie für den Test dieses Programms in der EXCEL-Tabelle einen mit Zahlen gefüllten Bereich brauchen. Falls Sie mehrere Funktionen aufrufen wollen, empfiehlt es sich die Zellen, in die Sie Funktionsergebnisse bringen wollen, vor dem Aufruf der Dialogbox durch entsprechende Texte zu kennzeichnen.

Programmliste:

```
'Deklarationen auf Modulebene
Option Explicit
Private OptNr As Integer

Private Sub UserForm_Initialize()
  OptSum.Value = True
  OptNr = 1
End Sub

Private Sub AZelle_Change()
  If AZelle.Value <> "" Then Range(AZelle.Value).Select
End Sub

Private Sub OptSum_Click()
  OptNr = 1
End Sub

Private Sub OptProd_Click()
  OptNr = 2
End Sub
```

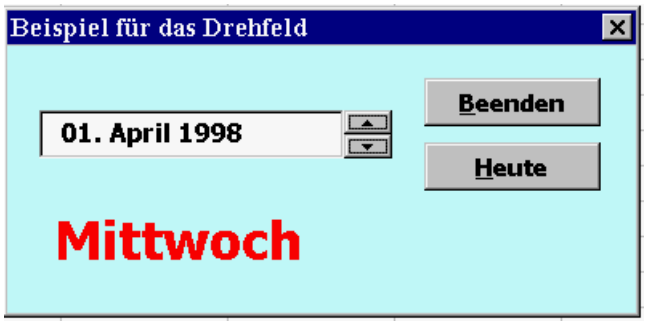
```
Private Sub OptAnz_Click()
  OptNr = 3
End Sub

Private Sub OptMin_Click()
  OptNr = 4
End Sub
:
:
:
Private Sub OptVar_Click()
  OptNr = 9
End Sub

Private Sub OKButton_Click()
  Dim bereich As String
  If AZelle.Value <> "" Then Range(AZelle.Value).Select
  If TBereich.Value = "" Or AZelle.Value = "" Then
    MsgBox "Bitte einen Tabellenbereich und eine Ausgabezelle angeben!"
    If TBereich.Value = "" Then
      TBereich.SetFocus
    Else
      AZelle.SetFocus
    End If
    Exit Sub
  End If
  bereich = TBereich.Value
  Select Case OptNr
    Case 1
      ActiveCell.Formula = "=SUM(" & bereich & ")"
    Case 2
      ActiveCell.Formula = "=PRODUCT(" & bereich & ")"
    Case 3
      ActiveCell.Formula = "=COUNT(" & bereich & ")"
    Case 4
      ActiveCell.Formula = "=MIN(" & bereich & ")"
    Case 5
      ActiveCell.Formula = "=MAX(" & bereich & ")"
    Case 6
      ActiveCell.Formula = "=AVERAGE(" & bereich & ")"
    Case 7
      ActiveCell.Formula = "=GEOMEAN(" & bereich & ")"
    Case 8
      ActiveCell.Formula = "=MEDIAN(" & bereich & ")"
    Case 9
      ActiveCell.Formula = "=VAR(" & bereich & ")"
  End Select
  AZelle.Value = ""
  AZelle.SetFocus
End Sub

Private Sub CancelButton_Click()
  Unload Me
End Sub
```

Drehfeld – Steuerelement (SpinButton) Beispiel: *Drehfeld.XLS*

Das Drehfeld ist eine vereinfachte Version der Bildlaufleiste. Es besitzt aber keinen Schieber, sondern es besteht nur aus zwei kleinen Schaltflächen. Jede der beiden hat als Symbol einen Pfeilkopf, der jeweils nach oben und nach unten gerichtet ist (oder links / rechts). Das Klicken auf eine der Schaltflächen erhöht bzw. verringert den internen Wert des Drehfeldes um den Betrag, der in der Eigenschaft ***SmallChange*** festgelegt ist. Die Eigenschaft ***LargeChange*** gibt es hier nicht. Der Wertebereich wird bei den Eigenschaften ***Min*** und ***Max*** festgelegt, seine Ausdehnung entspricht dem Datentyp für ganze Zahlen. Der Wert des Drehfeldes kann entweder über ein Textfeld angezeigt werden oder mit Hilfe der Eigenschaft ***ControlSource*** in eine Zelle im Excel-Tabellenblatt gebracht werden.

Beispiel: *Anzeige eines kleinen Kalenders*

Nach dem Start erscheint das aktuelle Datum mit Angabe des Wochentags. Über ein Drehfeld soll das Datum um einen Tag vor- bzw. zurückgesetzt werden. In einem Textfeld wird das Datum angezeigt, der Wochentag erscheint in einem darunterstehenden Bezeichnungsfeld. Die Schaltfläche **HEUTE** setzt das Datum wieder auf den aktuellen Tag zurück.



Erklärungen zum Datumssystem von EXCEL:

Excel verwendet intern für die Datumswerte eine numerische Darstellung, d.h. jedem Datum entspricht ein bestimmter Wert eines Bereiches. Im Menü

→ **EXTRA** → **OPTIONEN** → **BERECHNEN**

können die zwei verschiedenen Datumssysteme eingestellt werden:

System 1900	Anfang 1.1.1900 = 1	Ende: 31.12.9999 = 2958465
System 1904	Anfang 2.1.1904 = 1	Ende: 31.12.9999 = 2957003

Sie können sich die numerischen Werte direkt in Excel anzeigen lassen, indem Sie eine Zelle mit einem Datum als Zahl formatieren – z.B. erhalten Sie für den 1.April 1998 den Wert 35886 (im System 1900). Wenn Sie Werte ausserhalb der angegebenen Datumsbereiche eingeben z.B. 31.12.1899 so erhalten sie kein Datum, sondern Excel betrachtet Ihre Eingabe als Text.

Im vorliegenden Beispiel wurde bei den Eigenschaften des Drehfeldes ***Min*** auf 1, ***Max*** auf 2958465 und ***SmallChange*** auf 1 gesetzt.

Beschreibung der Prozeduren:

Die **Prozedur `SetValue`** wird von verschiedenen Stellen aus aufgerufen und jedesmal wird ein Datumswert in den CallByValue-Parameter übergeben. Die Prozedur wird beim Starten des Programms (Ereignis **`Initialze`** der **UserForm**), beim Klicken der Schaltfläche **HEUTE** (Ereignis **`Click`** der Schaltfläche **ToDayButton**) und beim Ändern des Drehfeldes (Ereignis **`Change`** des Drehfeldes SpinButton1) aufgerufen. Das übergebene Datum wird als Eigenschaft ***Text*** im Textfeld im Format tt-mmmm-jjjj angezeigt. Der Wochentag wird als Eigenschaft ***Caption*** im Bezeichnungsfeld Label1 angezeigt, wobei zunächst mit der Funktion **Weekday** der Wochentag aus dem übergebenen Datum extrahiert und dann über die Funktion **Format** in einen Text umgewandelt wird. Zuletzt muss der Eigenschaft ***Value*** des Drehfeldes das Datum als Zahl vom Typ Long (lange Ganzahl) zugewiesen werden.

In der **Prozedur `SpinButton1_Change`** wird zuerst der Wert des Drehfeldes durch die Funktion **Cdate** (Convert date) in das Datumformat konvertiert, bevor es der Prozedur **`SetValue`** als Parameter übergeben wird.

In der **Prozedur `ToDayButton`** wird die Funktion **Date**, die das Systemdatum zurückgibt, verwendet.

Programmliste:

```
Sub SetValue(ByVal datum As Date)
  TextBox1.Value = Format(datum, "dd. mmmm yyyy")
  Label1.Caption = Format(WeekDay(datum), "dddd")
  SpinButton1.Value = CLng(datum)
End Sub

Private Sub UserForm_Initialize()
  SetValue (Date)
End Sub

Private Sub SpinButton1_Change()
  SetValue (CDate(SpinButton1.Value))
End Sub

Private Sub TextBox1_Enter()
  MsgBox "Finger weg!"
End Sub

Private Sub ToDayButton_Click()
  SetValue (Date)
End Sub

Private Sub EndButton_Click()
  Unload Me
End Sub
```

Multiseiten – Steuerelement (MultiPage) Beispiel: *Multiseiten.XLS*

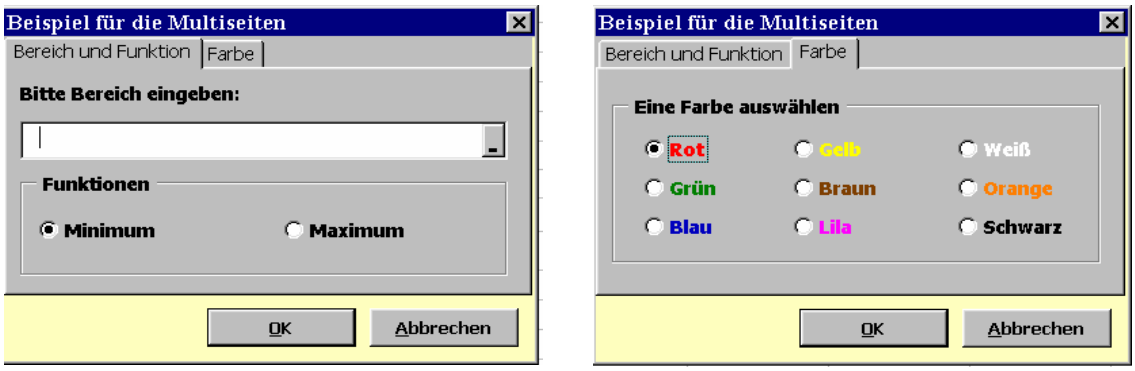
Das Multiseiten-Steuerelement ist eines der neuen Elemente in EXCEL ab Office 97. Die Multiseiten sind praktisch mehrere Fensteroberflächen, die über ein Register erreichbar sind. Jede Seite verfügt über verschiedene Steuerelemente, die Eigenschaften jeder Seite können unabhängig voneinander eingestellt werden. Standardmässig verfügt die Multiseite nach dem Einfügen in die UserForm über zwei Registerblätter Page1 und Page 2. Man kann aber über das Kontextmenü der Multiseiten weitere Seiten hinzufügen. Die Seiten werden mit einem Index (beginnend bei 0) durchnumeriert. Dieser Wert kann über die Eigenschaft ***Value*** abgefragt werden.

Beispiel: *Maximum / Minimum der einzelnen Zeilen einer Tabelle färben*

Bei grossen Tabellen mit Zahlen wird gewünscht, dass sich das Minimum und das Maximum der Zeilen farbig vom Rest unterscheidet.

Das Programm zeigt ein Fenster mit zwei Seiten an. In der ersten Seite wird der gewünschte Zahlenbereich über ein Referenzfeld angegeben und über die Optionsfelder die Funktion Minimum oder Maximum ausgewählt – als Standard wird Minimum vorgesehen. In der zweiten Seite wird die Farbe ausgesucht – als Standardfarbe wird Rot angenommen.

Darstellung der beiden Seiten:



Beschreibung der Prozeduren:

Die **Prozedur `SetColor`** benötigt einige Parameter: In ***bereich*** wird der vom Anwender markierte Bereich übergeben, in ***Fkt*** wird die Nummer der gesetzten Option 1 für Min und 2 für Max übernommen, die übrigen Parameter stellen die Farbwerte dar (z.B. Rot: 255, 0, 0).

Über die Eigenschaft ***Count*** wird die Anzahl Zeilen (Rows) und Spalten (Columns) des markierten Bereichs ermittelt. In einer äusseren **For-Schleife**, die der Reihe nach alle Zeilen durchläuft, wird das Minimum bzw. Maximum pro Zeile ermittelt und in einer inneren Schleife, die pro Zeile alle Spalten durchprüft, werden die entsprechenden Zahlen in der gewünschten Farbe gefärbt.

Erklärung zur Schreibweise **`Application.Min`**.... bzw. **`Application.Max`**....:
in VBA kann man auf dadurch auf die Tabellenfunktionen von Excel zugreifen, das Schlüsselwort **`Application`** bezeichnet die Excel-Anwendung selbst.

Diese Prozedur ist eine der klassischen Anwendungen der Excel-Programmierung, bei der oft markierte Bereiche durchlaufen werden müssen.

In der **Prozedur für die Schaltfläche OK** werden die Seiten ermittelt und es erfolgt der Wechsel der Seiten.
Wenn der Anwender keinen Bereich markiert hat und die Schaltfläche OK anklickt, erscheint die Aufforderung, einen Bereich anzugeben. Dann wird der Fokus auf das Referenzfeld gesetzt, um die Eingabe sofort zu ermöglichen. Da aber das Fenster zwei Seiten hat, kann man nicht sicher sein, ob die erste Seite, in der sich das Referenzfeld befindet, auch aktiv ist. Daher wird über die Eigenschaft ***Value*** des MultiPage-Elements festgestellt, welche Seite aktiv ist. Falls die Seite für die Farbe aktiv ist – also der Index auf 1 steht – muss er auf 0 verringert werden. Dann wird die **Prozedur `SetColor`** ausgeführt.

`UserForm_Initialize:` Hier werden die Startwerte für die Steuerelemente festgelegt: die Standardfunktion soll ***Minimum*** sein, die Standardfarbe ***Rot*** und die Startseite der Multiseiten soll die erste Seite sein. Anschliessend wird eine Tabelle mit Zahlen zum Ausprobieren des Dialogs aktiviert.

Programmliste:

```
' Deklarationen auf Modulebene
Private rot As Integer
Private gruen As Integer
Private blau As Integer
Private FktNr As Integer

Sub SetColor(bereich As Range, Fkt As Integer, rot As Integer, _
             gruen As Integer, blau As Integer)
  Dim Zeilen As Byte, Spalten As Byte
  Dim i As Byte, j As Byte, r As Byte
  Dim Zeilfkt As Double
  Zeilen = bereich.Rows.Count
  Spalten = bereich.Columns.Count
  For j = 1 To Zeilen
    If Fkt = 1 Then
      Zeilfkt = Application.Min(bereich.Rows(j))
    Else
      Zeilfkt = Application.Max(bereich.Rows(j))
    End If
    For r = 1 To Spalten
      If bereich.Rows(j).Columns(r).Value = Zeilfkt Then
        bereich.Rows(j).Columns(r).Font.Color = RGB(rot, gruen, blau)
      End If
    Next r
  Next j
End Sub

Private Sub UserForm_Initialize()
  Minimum.Value = True
  OptRot.Value = True
  MultiPage1.Value = 0
  ThisWorkbook.Worksheets("Multiseiten").Activate
End Sub

Private Sub Maximum_Click()
  FktNr = 2
End Sub
```

```
Private Sub Minimum_Click()
  FktNr = 1
End Sub

Private Sub OptRot_Click()
  rot = 255
  gruen = 0
  blau = 0
End Sub

Private Sub OptGruen_Click()
  rot = 0
  gruen = 255
  blau = 0
End Sub
:
:
Private Sub OptBraun_Click()
  rot = 128
  gruen = 64
  blau = 0
End Sub

Private Sub OptLila_Click()
  rot = 255
  gruen = 0
  blau = 128
End Sub
:
:
Private Sub OptSchwarz_Click()
  rot = 0
  gruen = 0
  blau = 0
End Sub

Private Sub OKButton_Click()
  Dim pagenr As Byte, bereich As String
  If RefEdit1.Value = "" Then
    MsgBox "Bitte Bereich markieren!"
    pagenr = MultiPage1.Value
    If pagenr = 1 Then MultiPage1.Value = pagenr - 1
    RefEdit1.SetFocus
    Exit Sub
  End If
  bereich = RefEdit1.Value
  SetColor Range(bereich), FktNr, rot, gruen, blau
End Sub

Private Sub CancelButton_Click()
  Unload Me
End Sub
```

Funktion RGB(rot,grün,blau)

Die Argumente rot ,grün, und blau sind vom Typ Variant (Integer) im Bereich von 0 bis 255

Farbe	*Rot*	*Grün*	*Blau*
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Braun	128	64	0
Lila	255	0	128
Orange	255	128	0
Gelb	255	255	0
Weiß	255	255	255

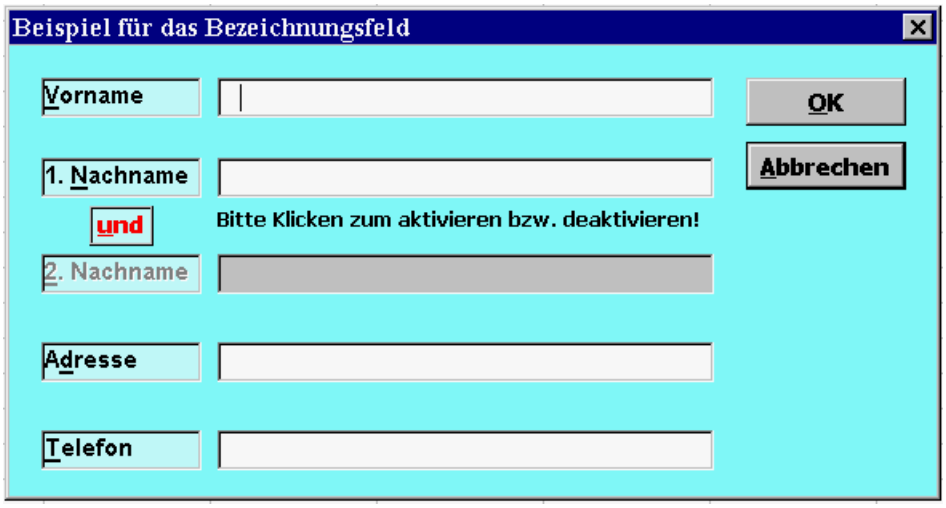
Bezeichnungsfeld – Steuerelement (Label) Beispiel: *Bezeichnungsfeld.XLS*

Das Bezeichnungsfeld-Steuerelement wird häufig nur zur Anzeige von Text in einem Dialogfeld verwendet. Der Text wird bei der Eigenschaft ***Caption*** angegeben. Das folgende Beispiel zeigt, dass man dieses Steuerelement auch zur Aktivierung von anderen Elementen verwenden kann.

Beispiel: *Steuerung der Eingabe im Formular für die Adresseingabe*

Das ist ein Musterbeispiel dafür, dass man Eingaben in eine Tabelle über ein Dialogfenster – man sagt dazu auch Eingabeformular machen kann. Die Erweiterung des Beispiels mit Routinen für die Angabe von Tabellenblättern und Bereichen, sowie für das Zurückschreiben ist zu ergänzen.

Die Dialogbox zeigt 5 Textfelder mit dazugehörigen Bezeichnungsfeldern:



Zwischen dem 1. und dem 2.Nachnamen befindet sich ein Bezeichnungsfeld mit dem Text "und". Zu Beginn sind der Text und Eingabefeld für den zweiten Nachnamen inaktiv. Ein Klick auf den Text "**und**" aktiviert die beiden Felder für die Eingabe des 2.Nachnamens.

Prozedur `Label6_Click`

Der Text und das Eingabefeld für den 2.Nachnamen werden über die Eigenschaft ***Enabled*** beim Anklicken auf ihr Gegenteil gesetzt. Wenn die Felder aktiv sind, soll der Hintergrund des Eingabefeldes weiss erscheinen und den Fokus bekommen, damit der Cursor im Feld blinkt. Wenn die Felder inaktiv sind, soll der Hintergrund grau sein. Die Farben werden mit der **Funktion RGB** gesetzt.

Prozedur `OKButton_Click`

Beim Klicken auf **OK** werden die gerade gemachten Eingaben am Bildschirm angezeigt und die Eingabefelder wieder gelöscht.

Programmliste:

```
Private Sub Label6_Click()
  NNText2.Enabled = Not NNText2.Enabled
  Nachname2 Enabled = Not Nachname2.Enabled
  If Nachname2.Enabled Then
    Nachname2.BackColor = RGB(255, 255, 255)
                                    'Wenn aktiv, dann Weiss
    Nachname2.SetFocus
  Else
    Nachname2.BackColor = RGB(180, 180, 180)
                                    'Wenn nicht aktiv, dann grau
  End If
End Sub


Private Sub OKButton_Click()
  Dim helptext As String
  Dim zneu As String
  zneu = Chr(13) & Chr(10)
  If Nachname2.Text <> "" Then helptext = "-" & Nachname2.Text
  msg = Vorname & " " & Nachname1
  msg = msg & helptext & "," & zneu
  msg = msg & Adresse & zneu
  msg = msg & Telefon
  MsgBox msg
  Vorname.Text = ""
  Nachname1.Text = ""
  Nachname2.Text = ""
  NNText2.Enabled = False
  Nachname2.Enabled = False
  Nachname2.BackColor = RGB(180, 180, 180)
  Adresse.Text = ""
  Telefon.Text = ""
  Vorname.SetFocus
End Sub


Private Sub CancelButton_Click()
  Unload Me
End Sub
```

Kontrollkästchen / Umschaltfeld – Steuerelemente (CheckBox / ToggleButton)

Beispiel: *Kontroll_Umschalt.XLS*

Zwischen den beiden Steuerelementen besteht in der Funktion kein Unterschied. Beide werden meist dort eingesetzt, wo man die Auswahl einer oder mehrerer Zusatzoptionen erlauben will. Im Unterschied zu Optionsfeldern können gleichzeitig mehrere aktiv sein.

Kontrollkästchen können drei Zustände annehmen:

- **Nicht aktiviert**: das Feld des Kontrollkästchens ist leer und weiss. Dies entspricht dem Zustand False, d.h. das Element wurde absichtlich deaktiviert.
- **Aktiviert**: Im weisen Feld ist ein Haken zu sehen. Dies entspricht dem Zustand True, d.h. das Element wurde gesetzt.
- **Abgeblendet**: das Feld erscheint grau schattiert und mit einem grauen Haken. Dieser dritte Zustand entspricht dem Wert NULL und kann im Dialogfeld nur erreicht werden, wenn die Eigenschaft ***TripleState*** auf True gesetzt ist oder wenn ihm von Programm aus der Wert **NULL** zugewiesen wurde.

Ein **Umschaltfeld** sieht rein optisch wie eine Befehlsschaltfläche aus, bleibt aber gesetzt wie ein Kontrollkästchen. Kontrollkästchen werden eher dann verwendet, wenn wenig Platz im Dialogfeld ist. Die Wahl zwischen Kontrollkästchen und Umschaltfeld ist hauptsächlich Geschmackssache. Übrigens ist das Umschaltfeld ein neues Element von EXCEL ab Office 97.

1.Beispiel: *Eingabe von Artikel, Preis und Anzahl mit Auswahl von zwei Werten für die Mehrwertsteuer und Übertragung in eine Tabelle. Verwendung eines Kontrollkästchens.*

Bei jedem Klicken auf die Schaltfläche **OK** werden die Eingaben in eine Tabelle, die beim Programmstart neu angelegt wird, übertragen. Für jeden Artikel werden sowohl der Bruttopreis (Preis plus Mehrwertsteuer) als auch der Gesamtpreis (Bruttopreis mal Anzahl) berechnet. Für die Mehrwertsteuer wird immer 15 % angenommen, es sei denn, der Anwender wählt im Dialogfeld das Kontrollkästchen "Nur 7% MWSt"

Da das Dialogfeld mehr als nur eine Eingabe erlauben soll, muss es so lange, wie der Anwender es wünscht, geöffnet bleiben. Es kann nur über die Schaltfläche **ABBRECHEN** geschlossen werden.

Beschreibung der Prozeduren:

Die **Prozedur `Tabellenkopf`** wurde als Makro aufgezeichnet und "ausgemistet". Sie fügt eine neue Tabelle in die aktuelle Mappe ein, generiert die Beschriftungen und formatiert die letzte Spalte als Prozent. Dann geht der Cursor in Zelle A2, von da aus werden dann die Daten eingetragen.

Funktion `MWSt:` Wenn das Kontrollkästchen gesetzt ist, werden nur 7% berechnet, sonst 15%. Dem Wert des Kontrollkästchens entsprechend wird der Rückgabewert gesetzt.

In der **Prozedur `InitDialog`** wird das Kontrollkästchen auf False gebracht, da es vor einer Eingabe nicht gesetzt sein soll. Ausserdem werden alle Textfelder des Dialogs geleert.

Bei den Textfeldern für Preis und Anzahl wird in den **Prozeduren `Anzahl_Change`** und **`NPreis_Change`** das eingegebene Zeichen zunächst in eine Stringvariable gebracht und es wird kontrolliert, ob es sich um eine Ziffer handelt. Die Kontrolle des Minuszeichens ist etwas problematisch, da dieses Zeichen nicht zu Beginn einer Zahl z.B.-3, sondern nur am Ende der Zahl d.h. 3- akzeptiert wird. Beachten Sie die Möglichkeit mit Stringfunktion **`Left$`** das zuletzt eingegebene Zeichen abzuschneiden, also aus dem Textfeld zu entfernen.

Prozedur für das **Click-Ereignis** von **OK:** Beim ersten Aufrufen dieser Prozedur befindet sich der Cursor in Zelle A2. Beachten Sie, dass für die Übertragung in die einzelnen Zellen, die Methode ***Offset*** verwendet wird, bei der in Klammern die Parameter für die Verschiebung angegeben werden. Die erste Zahl ist für die Zeile, die zweite für die Spalte. Der Wert 0 bedeutet, dass die Markierung in der gleichen Zeile oder Spalte bleiben soll. Ein positiver Wert bewirkt eine Bewegung nach rechts für eine Zeile, nach unten für eine Spalte. Ein negativer Wert schiebt den Cursor in einer Zeile nach links, in einer Spalte nach oben.
Die Prozedur ist auch mit einer Fehlerbehandlungsroutine ausgestattet um Programmabbrüche zu vermeiden, falls der Anwender eine fehlerhafte oder unvollständige Eingabe machen sollte.

Programmliste:

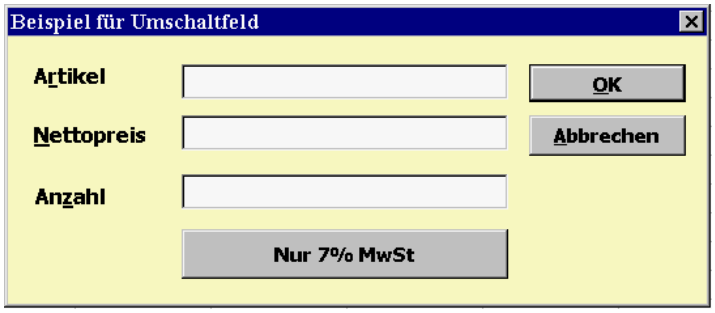
```
Sub Tabellenkopf()
      ' Diese Prozedur wurde als Makro aufgezeichnet
  Sheets.Add
  Range("A1") = "Artikel"
  Range("B1") = "Preis"
  Range("C1") = "Anzahl"
  Range("D1") = "Brutto"
  Range("E1") = "Gesamt"
  Range("F1") = "MwSt"
  Columns("F:F").Select
  Selection.NumberFormat = "0.00%"
  Range("A1:F1").Select
  Selection.Interior.ColorIndex = 48
  Selection.Font.ColorIndex = 2
  With Selection.Font
    .Name = "Arial"
    .Size = 14
    .ColorIndex = 2
  End With
  Range("A2").Select
End Sub

Private Sub UserForm_Initialize()
  Tabellenkopf
End Sub
```

```
Function mwst() As Single
  If mwst7.Value = True Then
    mwst = 0.07
  Else
    mwst = 0.15
  End If
End Function

Sub InitDialog()
  Artikel = ""
  NPreis = ""
  Anzahl = ""
  mwst7.Value = False
  Artikel.SetFocus
End Sub

Private Sub Anzahl_Change()
  Dim Eingabe As String
  Eingabe = Anzahl.Value
  If Eingabe = "-" Then
    MsgBox "Bitte nur positive Zahlen eingeben!"
    Anzahl.Value = Left$(Eingabe, Len(Eingabe) - 1)
    Exit Sub
  End If
  If Not IsNumeric(Eingabe) And Eingabe <> "" Then
    MsgBox "Bitte nur Zahlen eingeben!"
    Anzahl.Value = Left$(Eingabe, Len(Eingabe) - 1)
  End If
End Sub

Private Sub NPreis_Change()
  Dim Eingabe As String
  Eingabe = NPreis.Value
  If Eingabe = "-" Then
    MsgBox "Bitte nur positive Zahlen eingeben!"
    NPreis.Value = Left$(Eingabe, Len(Eingabe) - 1)
    Exit Sub
  End If
  If Not IsNumeric(Eingabe) And Eingabe <> "" Then
    MsgBox "Bitte nur Zahlen eingeben!"
    NPreis.Value = Left$(Eingabe, Len(Eingabe) - 1)
  End If
End Sub

Private Sub OKButton_Click()
  Dim Brutto As Currency
  Dim Gesamt As Currency
  Dim MwstSatz As Single
  On Error GoTo KontrollError
  If Anzahl.Value <= 0 Or NPreis.Value <= 0 Then
    MsgBox "Preis oder Anzahl negativ oder Null - neu eingeben!"
    Exit Sub
  End If
  ActiveCell.FormulaR1C1 = Artikel.Value
  ActiveCell.Offset(0, 1).Activate
  ActiveCell.FormulaR1C1 = CCur(NPreis.Value)
  ActiveCell.Offset(0, 1).Activate
```

```
    ActiveCell.FormulaR1C1 = Anzahl.Value
    ActiveCell.Offset(0, 1).Activate
    MwstSatz = mwst()
    Brutto = CCur((NPreis.Value * MwstSatz) + NPreis.Value)
    ActiveCell.FormulaR1C1 = Brutto
    ActiveCell.Offset(0, 1).Activate
    Gesamt = CCur(Brutto * Anzahl.Value)
    ActiveCell.FormulaR1C1 = Gesamt
    ActiveCell.Offset(0, 1).Activate
    ActiveCell.FormulaR1C1 = MwstSatz
    ActiveCell.Offset(1, -5).Activate
    InitDialog
    Exit Sub
KontrollError:        ' Wenn die Eingabe nicht vollständig ist
    If Err = 13 Then
      msg = "Sie haben keine Eingabe gemacht!"
    Else
      msg = "Ein Fehler mit dem Code " & CStr(Err) & " ist aufgetreten!"
    End If
    msg = msg & Chr$(13) & Chr$(10) & "Programmende!"
    MsgBox msg
    Unload Me
End Sub

Private Sub CancelButton_Click()
    Unload Me
End Sub
```

2. Beispiel *Beispiel 1 mit Verwendung eines Umschaltfeld-Steuerelements*

Um die Ähnlichkeit zwischen den beiden erwähnten Steuerelementen zu zeigen, wurde das vorherige Beispiel an das Umschaltfeld angepasst. Das Steuerelement erhält den gleichen Namen, so dass das Programm unmittelbar übertragbar ist.



3. Beispiel *Beispiel aus der Hilfe*

Wenn Sie bei den Eigenschaften eines Kontrollkästchens oder eines Umschaltfeldes auf die Eigenschaft ***TripleState*** gehen und dann **F1** drücken, um die Hilfe zu aktivieren, finden Sie das Beispiel **TripleState – Eigenschaft**. Dieses Beispiel zeigt Ihnen, wie in einer Dialogbox gleichzeitig beide Steuerelemente verwendet werden und Sie können daraus auch entnehmen, wie sich die Beschriftung der Steuerelemente über die Eigenschaft ***Caption*** an den aktuellen Zustand anpassen lässt.

Listenfeld / Kombinationsfeld – Steuerelemente (ListBox / ComboBox)

Beispiel: *Listen_Kombofeld.XLS*

Listenfeld – Steuerelement

Das Listenfeld bietet die Möglichkeit aus einer Liste einen oder mehrere Einträge auszuwählen. Das Listenfeld ist dem Kombinationsfeld in Funktion und Behandlung sehr ähnlich. Die beiden Elemente unterscheiden sich optisch voneinander und es ist vom Geschmack und auch vom vorhandenen Platz in dem Dialogfeld abhängig, welches Steuerelement verwendet wird.

Das Listenfeld ist ein rechteckiges Feld, in dem eine Auflistung zu sehen ist. Wenn die Liste länger als das Feld ist, erscheint automatisch eine Bildlaufleiste am rechten Rand des Feldes.

Das Kombinationsfeld sieht aus wie ein Textfeld, das bei Bedarf mit einem Klick auf den Pfeilkopf, der sich rechts im Feld befindet, aufgeklappt werden kann. Dadurch wird eine Liste angezeigt, aus der man einen Eintrag auswählen kann. Wenn die Liste länger als der zur Verfügung stehender Platz ist, erscheint auch hier automatisch eine Bildlaufleiste.

Zum Unterschied vom Kombinationsfeld ist beim Listenfeld neben der einfachen Auswahl auch mehrfache und erweiterte Auswahl möglich. Die Art der Auswahl kann über die Eigenschaft ***MultiSelect*** gesteuert werden oder wie im folgenden Beispiel gezeigt wird, dynamisch während der Laufzeit gesetzt werden.

Einfache Auswahl: Es kann nur ein Eintrag aus der Liste ausgewählt werden, eine erneute Wahl löscht die Markierung der vorherigen Wahl. Die Programmierung erfolgt hier über die Standardeigenschaft ***Value*** und das Standardereignis ***Click***. Die Eigenschaft ***Multiselect*** wird auf **0** oder auf die VBA-Konstante ***frmMultiSelectSingle*** gesetzt.

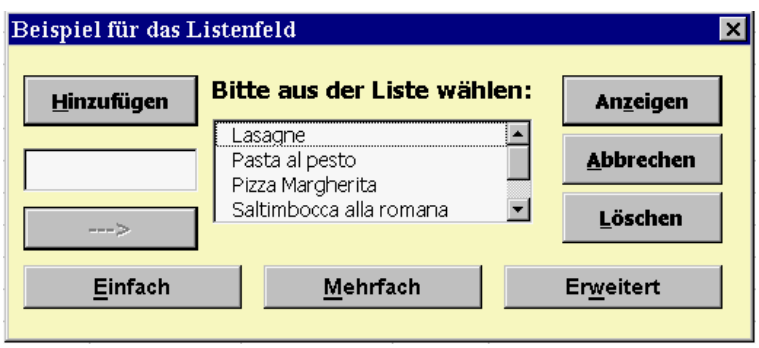
Mehrfache Auswahl: Die Markierung verschiedener - auch nicht zusammenhängender Einträge - ist erlaubt. Ein Klick auf einen Eintrag reicht aus, um ihn auszuwählen. Die Markierung weiterer Elemente löscht die vorherigen Markierungen nicht. Um die Markierung eines Eintrags zu entfernen, muss noch einmal auf den Eintrag geklickt werden.

Erweiterte Auswahl: Diese Auswahl wird verwendet, wenn die einfache Auswahl die Regel ist, aber auch die mehrfache Auswahl erlaubt sein soll. Wie bei einer einfachen Auswahl löscht eine erneute Auswahl die Markierung der vorherigen Wahl. Mit Hilfe der Steuertasten Ctrl und Shift ist aber auch hier eine mehrfache Auswahl von nicht zusammenhängenden oder zusammenhängenden Einträgen möglich - ähnlich wie im Windows-Explorer.

Beispiel:

Aus einer Liste von italienischen Spezialitäten kann man eine Auswahl treffen. Die Art der Auswahl kann gewählt werden. Es können auch Spezialitäten neu hinzugefügt werden bzw. gelöscht werden. Dieses Beispiel zeigt die Manipulation von Daten, wie sie auch für Datenbanken von grundlegender Bedeutung ist.

Abbildung:



Die Schaltflächen haben folgende Bedeutung:

Über die drei Schaltflächen: **EINFACH / MEHRFACH / ERWEITERT** kann die Art der Auswahl bestimmt werden.

Die Schaltfläche **ANZEIGE** zeigt die vom Anwender markierte Einträge in einer MessageBox an. Diese Ausgabe am Bildschirm dient nur zu Demonstrationszwecken, in einem "wirklichen" Programm werden die gewählten Einträge natürlich anders bearbeitet.

Über die Schaltfläche **LÖSCHEN** können beliebig viele Einträge aus der Liste gelöscht werden. Die Schaltfläche **HINZUFÜGEN** aktiviert das darunterstehende Eingabefeld, in dem ein Text eingegeben werden kann. Der Text wird dann über die Schaltfläche mit dem Pfeil in die Liste aufgenommen.

Beschreibung der Prozeduren:

Die Liste wird mit der **Prozedur `ListeBilden`** gefüllt, die beim Laden des Dialogs – Ereignis Initialize – ausgeführt wird. Die Liste wird aus einer Tabelle gelesen. In einer **`Do Until ... Loop`** werden ab A1 in Spalte A die Zellen mit der ***Formula***-Eigenschaft überprüft, ob ein Eintrag vorhanden ist. Diese Einträge werden über die ***AddItem***-Methode dem Listenfeld zugewiesen. Die Schleife wird solange wiederholt, bis eine der Zellen in der Spalte A leer ist.

In der Programmliste finden Sie ausserdem in Kommentarzeilen, die beiden anderen Möglichkeiten eine Liste zu füllen: *Liste ohne Tabelle füllen* und Liste mit Hilfe der ***RowSource*** – Eigenschaft mit einer *Tabelle verknüpfen*.

Die **Prozedur `ListeInTabelleZurück`** schreibt beim **Schliessen des Dialogfeldes** die komplette Liste in die Tabelle zurück. Das Zurückschreiben erfolgt mit Hilfe einer **`For`** - Schleife, wobei über die Eigenschaft ***Formula*** der Eintrag der Liste in die Zelle geschrieben wird. Die Zellenadresse wird mit Hilfe des Zählers der Schleife gebildet. Beachten Sie, dass der Zeilenindex der Liste mit **0** beginnt.

Wenn die Liste im Dialogfeld nicht geändert werden dürfte, wäre das Zurückschreiben nicht notwendig, da aber in diesem Beispiel Änderungen zugelassen sind, muss die Liste in der Tabelle immer aktualisiert werden. Falls die neue Liste gleich lang oder länger ist als die alte Liste, läuft alles problemlos. Wenn die neue Liste aber kürzer ist, bleiben am Ende alte Einträge übrig. Diese werden durch eine **`Do Until ... Loop`** am Ende der Prozedur gelöscht. Dazu wird die Zelle nach dem letzten zurückgeschriebenen Eintrag überprüft, ob sie leer ist. Wenn sie leer ist, passiert nichts, falls sie nicht leer ist, wird der Inhalt gelöscht. Das Löschen der Zelle wird bis zur ersten leeren Zelle fortgesetzt. Man könnte auch vor dem Zurückschreiben zuerst die alte Liste in der Tabelle löschen!

Die drei **Prozeduren** für die Click – Ereignisse für die Art der Auswahl finden Sie unter: **`einfach_Click, erweitert_Click`** und **`merfach_Click`**.

Die **Prozedur `ShowButton_Click`** wird beim Drücken der Schaltflächen Anzeigen ausgeführt.
Die Stringvariablen **`msg`** und **`zneu`** werden für die Ausgabe in der MessageBox benötigt, wobei **`zneu`** als Abkürzung für den Zeilenumbruch – also neue Zeile in der MessageBox – verwendet wird. Die Variable **`msg`** enthält entweder alle markierten Listeneintragungen oder entsprechende Texte, falls kein Eintrag markiert wurde bzw. die Liste leer ist.

Über die ***ListCount***-Eigenschaft erhalten wir die gesamte Anzahl der Einträge im Listenfeld. Falls die Liste keine Einträge enthält, wird die Zahl Null zurückgegeben und es wird eine entsprechende Meldung angezeigt. Da auch mehrfache und erweiterte Auswahl möglich ist, wird anschliessend ermittelt, welche Einträge der Anwender ausgewählt hat. Diese Untersuchung erfolgt in einer **`For`** - Schleife. Da die Numerierung der Listenfelder mit **0** beginnt, muss der Zähler bei z.B. 5 Einträgen von 0 bis 4 gehen. Falls der Anwender einen Eintrag in der Liste markiert hat, besitzt die ***Selected***-Eigenschaft den Wert ***true***, sonst ***false***. Die ***List***-Eigenschaft gibt den Text des markierten Eintrages zurück und es wird zusätzlich die Nummer des Listenelementes angezeigt. Diese Angaben werden für alle markierten Elemente in der Stringvariablen **`msg`** zwischengespeichert.
Die Boolesche Variable **`control`** wird auf ***true*** gesetzt, falls mindestens ein Eintrag markiert wurde. Sie wird zur Steuerung der Anzeigemeldung verwendet.

Die **Prozedur `DelButton_Click`** soll das Löschen von allen markierten Einträgen bewirken. Beim Klicken auf die Schaltfläche **LÖSCHEN** werden alle Einträge, die nicht markiert sind, in ein neues Datenfeld mit dem Namen **`merke`** kopiert. Die Dimension des Datenfeldes wird mit der Anweisung **`ReDim`** dynamisch bestimmt, d.h. es werden soviel Felder für das Datenfeld reserviert, wie Listeneinträge vorhanden sind. Nachdem das Kopieren beendet ist, wird die gesamte Liste der **`ListBox`** mit der Methode ***Clear*** gelöscht. Die Liste wird mit Methode ***AddItem*** aus den im Datenfeld geretteten Einträgen neu aufgebaut. Eine Alternative wäre das Löschen von einzelnen Einträgen mit der Methode ***RemoveItem***.

Die **Prozedur `NewButton_Click`** dient zum Hinzufügen neuer Eintragungen in die Liste. Die Schaltfläche kann zwei Beschriftungen haben: **HINZUFÜGEN** oder **SPERREN**. Die Prozedur zeigt die nötigen Anweisungen für diese beiden Fälle:
Für den Fall Hinzufügen werden das Textfeld für Eingabe ***EingabeBox*** und die Schaltfläche mit dem Pfeil aktiviert, das bedeutet die Eigenschaft ***Enabled*** der beiden Steuerelemente wird auf ***true*** gesetzt, die Beschriftung der Schaltfläche **HINZUFÜGEN** wird auf **SPERREN** geändert und der Focus wird auf das Textfeld für die Eingabe des neuen Eintrags gesetzt. Für den Fall SPERREN müssen Eingabefeld und Schaltfläche mit dem Pfeil inaktiviert werden und die Beschriftung muss wieder auf HINZUFÜGEN geändert werden.

Die **Prozedur `Pfeil_Click`** realisiert das Hinzufügen. Der Eintrag im Textfeld also **`EingabeBox.Value`** wird mit der ***AddItem***-Methode am Ende der Liste angehängt. Anschliessend wird das Eingabefeld entleert und der Focus wieder auf das Eingabefeld gesetzt.

Programmliste:

```
Sub ListeBilden()
  Dim i As Byte, zelle As String
       ' Liste ohne Tabelle füllen
       ' *************************
         ' Listbox1.AddItem "Spaghetti Carbonara"
         ' Listbox1.AddItem "Lasagne"
         ' Listbox1.AddItem "Saltimbocca alla Romana"
         ' Listbox1.AddItem "Tiramisu"
         ' Listbox1.AddItem "Pasta al Pesto"
         ' Listbox1.AddItem "Pizza Margherita"

       ' Liste mit einer Tabelle verknüpfen
       ' **********************************
         ' Listbox1.RowSource = "Tabelle1!A1:A6"
```

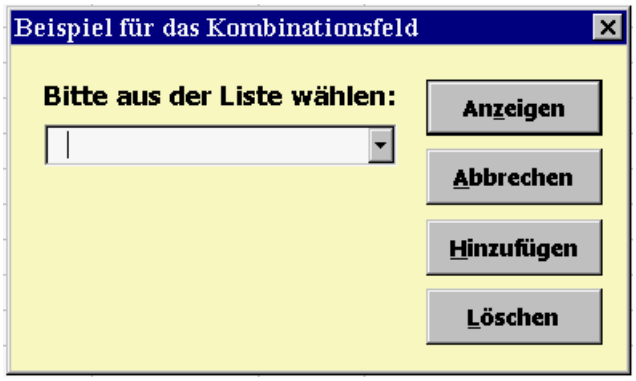
```
        ' Liste aus einer Tabelle auslesen,
        ' ohne sie mit der Tabelle zu verknüpfen
        ' ************************************
  i = 1
  zelle = "Tabelle1!A" & CStr(i)
  Do Until Range(zelle).Formula = ""
    Listbox1.AddItem Range(zelle).Formula
    i = i + 1
    zelle = "Tabelle1!A" & CStr(i)
  Loop
End Sub

Sub ListeInTabelleZurück()
  Dim i As Byte, zelle As String
  For i = 0 To Listbox1.ListCount - 1
    zelle = "Tabelle1!A" & CStr(i + 1)
    Range(zelle).Formula = Listbox1.List(i)
  Next i
  i = i + 1
  zelle = "Tabelle1!A" & CStr(i)
  Do Until Range(zelle).Formula = ""
    Range(zelle).Formula = ""
    i = i + 1
    zelle = "Tabelle1!A" & CStr(i)
  Loop
End Sub

Private Sub UserForm_Initialize()
  ListeBilden
  EingabeBox.Enabled = False
  Pfeil.Enabled = False
End Sub

Private Sub einfach_Click()
  Listbox1.MultiSelect = fmMultiSelectSingle
End Sub

Private Sub erweitert_Click()
  Listbox1.MultiSelect = fmMultiSelectExtended
End Sub

Private Sub merfach_Click()
  Listbox1.MultiSelect = fmMultiSelectMulti
End Sub

Private Sub ShowButton_Click()
  Dim zneu As String, msg As String
  Dim i As Byte, control As Boolean
  control = False
  zneu = Chr(13) & Chr(10
  msg = "Die Markierung ist auf:" & zneu
  If Listbox1.ListCount > 0 Then
    For i = 0 To Listbox1.ListCount - 1
      If Listbox1.Selected(i) Then
        msg = msg & "Nr: " & i & ", " & Listbox1.List(i) & zneu
        control = True
      End If
    Next i
```

```
    If control Then
      MsgBox msg
    Else
      MsgBox "Kein Eintrag wurde markiert!"
    End If
  Else
    MsgBox "Die Liste ist leer!"
  End If
End Sub

Private Sub DelButton_Click()
  Dim i As Byte, n As Byte, merke() As String
  If Listbox1.ListCount > 0 Then
    n = Listbox1.ListCount - 1
    ReDim merke(n)
    For i = 0 To Listbox1.ListCount - 1
      If Not Listbox1.Selected(i) Then merke(i) = Listbox1.List(i)
    Next i
    Listbox1.Clear
    For i = 0 To n
      If merke(i) <> "" Then Listbox1.AddItem merke(i)
    Next i
  Else
    MsgBox "Die Liste ist leer!"
  End If
End Sub

Private Sub NewButton_Click()
  If NewButton.Caption = "Hinzufügen" Then
    EingabeBox.Enabled = True
    Pfeil.Enabled = True
    NewButton.Caption = "Sperren"
    NewButton.Accelerator = "S"
    EingabeBox.SetFocus
  Else
    EingabeBox.Enabled = False
    Pfeil.Enabled = False
    NewButton.Caption = "Hinzufügen"
    NewButton.Accelerator = "H"
  End If
End Sub

Private Sub Pfeil_Click()
  Listbox1.AddItem EingabeBox.Value
  EingabeBox.Value = ""
  EingabeBox.SetFocus
End Sub

Private Sub CancelButton_Click()
  ListeInTabelleZurück
  Unload Me
End Sub
```

Kombinationsfeld

Da Listenfeld und Kombinationsfeld viele Gemeinsamkeiten aufweisen, wird als Illustrationsbeispiel für das Kombinationsfeld das gleiche Beispiel wir für das Listenfeld gezeigt. Dadurch können sehr gut auch die Unterschiede zwischen diesen beiden Steuerelementen gezeigt werden.

Im Unterschied zum Listenfeld ist bei dem Kombinationsfeld weder eine mehrfache noch eine erweiterte Auswahl möglich. Daher fallen die drei Schaltflächen:
EINFACH MEHRFACH ERWEITERT weg.
Der grösste Unterschied zum Listenfeld ist aber der, dass das Kombinationsfeld die Eingabe von Werten erlaubt. Im Beispiel für das Listenfeld mussten wir das Listenfeld um ein Textfeld erweitern, damit wir neue Einträge hinzufügen konnten. Dies fällt hier weg. Das Kombinationsfeld benötigt nicht diesen Umweg, da es eigentlich aus einem Listenfeld und aus dem Textfeld besteht (daher auch der Name).



Beschreibung der Prozeduren:

Die Liste des Kombinationsfeldes wird nach dem gleichen System wie im vorherigen Programm gebildet. Die Werte werden aus einer Tabelle ausgelesen – ohne Verknüpfung der Tabelle. Beim Schliessen des Dialogfeldes wird die Liste dann in die Tabelle zurückgeschrieben. Die **Prozeduren `ListeBilden`** und **`ListeInTabelleZurück`** zeigen ausser der Änderung von **ListBox** in **ComboBox** keinen Unterschied.

Die Programmierung des Kombinationsfeldes unterscheidet sich vom Listenfeld lediglich durch die Art der Erfassung - der Wahl des Anwenders: wenn ein Eintrag in der Liste durch einen Klick gewählt wird, schliesst sich die Liste und der Eintrag erscheint im Textfeld des Kombinationsfeldes. Man muss dann nur die Eigenschaft ***Value*** oder ***Text*** des Kombinationsfeldes abfragen wie es in der **Prozedur `ShowButton_Click`** zu sehen ist. Diese Prozedur ist viel einfacher, da hier keine mehrfache oder erweiterte Markierung möglich ist.
Vor der Anzeige wird überprüft, ob die Liste des Kombinationsfeldes überhaupt noch Einträge enthält. Dies ist hier notwendig, weil der Anwender die Möglichkeit hat, Einträge der Liste über die Schaltfläche **LÖSCHEN** zu löschen. Wenn die Liste leer wäre, würde der Zugriff auf **`ComboBox1.Value`** zum Programmabbruch führen.

Die **Prozedur `ComboBox1_Click`** weist der globalen Variablen **`EintragNr`** einen Eintrag zu, wenn der Anwender einen Eintrag aus der Liste auswählt oder neu hinzufügt.

Die **Prozedur `DelButton_Click`** zum Löschen ist hier einfach, da nur ein Eintrag markiert worden sein kann. Das **LÖSCHEN** findet nur statt, wenn die Liste der Einträge nicht leer ist und wenn ein Element mit einem Klick aus der Liste gewählt worden ist. Die Variable

EintragNr ist global und wird durch die **Prozedur ComboBox1_Click** gesetzt. Das Löschen wird mit der Methode ***RemoveItem*** realisiert. Nach dem Löschen wird der Text des Textfeldes gelöscht.

Das **HINZUFÜGEN** eines neuen Eintrags erfolgt so, dass man das Textfeld anklickt, den aktuellen Inhalt löscht und einen neuen Inhalt eintippt. Die **Prozedur NewButton_Click** zum **HINZUFÜGEN** enthält am Anfang eine Überprüfung, ob im Textfeld ein Text vorhanden ist. Nur dann passiert etwas, sonst wird nichts gemacht. Wenn die Liste Einträge enthält – es ist denkbar, dass der Benutzer alles gelöscht hat – wird zuerst überprüft, ob der einzufügende Text in der Liste vorhanden ist. Mit einer For-Schleife werden alle Listenelemente durchgeprüft. Wenn der neue Text gleich einem vorhandenen ist, wird die logische Variable **control** auf ***true*** gesetzt. In diesem Fall wird eine Meldung angezeigt. Nur wenn der Eintrag neu ist, wird er am Ende der Liste mit der ***AddItem***-Methode angefügt. Falls die Liste keine Einträge hat, wird der neue Text sofort eingefügt.

Programmliste:

```
Private EintragNr As Integer

Sub ListeBilden()
  Dim i As Byte, zelle As String
  i = 1
  zelle = "Tabelle1!A" & CStr(i)
  Do Until Range(zelle).Formula = ""
    ComboBox1.AddItem Range(zelle).Formula
    i = i + 1
    zelle = "Tabelle1!A" & CStr(i)
  Loop
End Sub

Sub ListeInTabelleZurück()
Dim i As Byte, zelle As String
  For i = 0 To ComboBox1.ListCount - 1
    zelle = "Tabelle1!A" & CStr(i + 1)
    Range(zelle).Formula = ComboBox1.List(i)
  Next i
  i = i + 1
  zelle = "Tabelle1!A" & CStr(i)
  Do Until Range(zelle).Formula = ""
    Range(zelle).Formula = ""
    i = i + 1
    zelle = "Tabelle1!A" & CStr(i)
  Loop
End Sub

Private Sub UserForm_Initialize()
  ListeBilden
End Sub

Private Sub ShowButton_Click()
  If ComboBox1.ListCount > 0 Then
    If ComboBox1.Text <> "" Then MsgBox ComboBox1.Value
  Else
    MsgBox "Die Liste ist leer!"
  End If
End Sub
```

```
Private Sub ComboBox1_Click()
  EintragNr = ComboBox1.ListIndex
End Sub

Private Sub DelButton_Click()
  If ComboBox1.ListCount > 0 Then
    If ComboBox1.Text <> "" Then
      ComboBox1.RemoveItem (EintragNr)
      ComboBox1.Text = ""
    Else
      MsgBox "Es wurde keinen Eintrag gewählt!"
    End If
  Else
    MsgBox "Die Liste ist leer!"
  End If
End Sub

Private Sub NewButton_Click()
  Dim i As Byte, control As Boolean
  If ComboBox1.Text <> "" Then
    If ComboBox1.ListCount > 0 Then
      For i = 0 To ComboBox1.ListCount - 1
        If ComboBox1.List(i) = ComboBox1.Text Then control = True
      Next i
      If control Then
        MsgBox "Der Eintrag ist schon vorhanden!"
      Else
        ComboBox1.AddItem ComboBox1.Text
        EintragNr = ComboBox1.ListCount - 1
      End If
    Else
      ComboBox1.AddItem ComboBox1.Text
      EintragNr = ComboBox1.ListCount - 1
    End If
  End If
End Sub

Private Sub CancelButton_Click()
  ListeInTabelleZurück
  Unload Me
End Sub
```

Register – Steuerelement (TabStrip) Beispiel: *Register.XLS*

Das Register – Steuerelement ist auch eine der Neuheiten von EXCEL ab Office 97. Wegen seiner äusseren Ähnlichkeit kann es mit dem Multiseiten – Steuerelement verwechselt werden. Die Funktionen der beiden Steuerelemente unterscheiden sich aber wesentlich voneinander.

Ein Multiseiten – Steuerelement stellt so viele verschiedene Dialogboxen zur Verfügung wie es Seiten umfasst, jede Seite hat ihre eigenen Steuerelemente. Ein Register besteht nur aus einer einzigen Dialogbox, obwohl es auch über Registerreiter verfügt. Egal, welchen Registerreiter man anklickt, es sind immer alle anderen Steuerelemente sichtbar. Dem Programmierer wird hier die Aufgabe gestellt, für die Änderung des Inhalts der Steuerelemente beim Wechseln der Registerreiter zu sorgen.

Standardmässig verfügt es über zwei Registerreiter. Die Zugriffe auf die verschiedenen Register erfolgen hier nicht über das Eigenschaftenfenster – wie bei Multiseiten – sondern nur über ein Programm. Auch das Benennen der Register kann nicht im Eigenschaftenfenster gemacht werden, sondern wird über das Kontextmenü des Registers durchgeführt. Die einzelnen Elemente eines Registers sind durchnumeriert, die Numerierung beginnt bei Null und kann im Programm über die Eigenschaft ***Value*** abgefragt werden.

Beispiel: *Ausgaben für den PC laufend ergänzen, Anzeige aktueller Summen*

Als Datenbasis dient eine Tabelle, die in Tabelle3 gespeichert ist und folgendes Aussehen hat:

	A	B	C	D	E
1	**Artikel**	**Preis**	**Menge**	**Datum**	**Gruppe**
2	Diskette	10.50 sFr	1	02.03.1995	Zubehör
:					
30	CDs	12.95 sFr	10	03.02.1997	Zubehör
31	Festplatte	500.00 sFr	1	05.02.1997	Hardware
32	Office 97	399.00 sFr	1	05.02.1997	Software
33	Papier	12.50 sFr	2	03.03.1997	Zubehör
34	Drucker	299.00 sFr	1	04.03.1997	Hardware
35	Monitor	1'575.95 sFr	1	04.04.1997	Hardware
36	Diskette	7.00 sFr	2	05.04.1997	Zubehör
:					

In einem Dialogfeld werden die Daten nach ihrer Gruppenzugehörigkeit – also Hardware, Software, Zubehör bzw. Gesamtsumme – summiert angezeigt.

Beschreibung der Prozeduren:

Im Deklarationsteil der UserForm werden die Variablen **`Blatt`** und **`Endzeile`** deklariert. Die Variable **`Blatt`** soll den Bezug zum Tabellenblatt herstellen und **`Endzeile`** enthält die Zeilennummer der letzten belegten Zeile der Liste.

In der **Prozedur `Initialize`** wird das Tabellenblatt zugewiesen und aktiviert. Bei einer Objekt-Zuweisung muss immer die ***Set***-Anweisung verwendet werden. Die ***Sheets***-Auflistung bezeichnet die Menge der Blätter der aktuellen Mappe. Die Methode ***Activate*** wechselt automatisch zu dem angegebenen Tabellenblatt. Statt der festen Zuordnung "Tabelle3" könnte man ein Referenzfeld für die Eingabe des Tabellenblattes verwenden. Anschliessend wird die **Prozedur `Listeermitteln`** aufgerufen, um die Länge der Liste zu ermitteln und als Standard wird der Reiter Gesamt gesetzt.

Die **Prozedur `Listeermitteln`** dient zur Ermittlung der Länge der Liste.
In einer **`Do Until ... Loop`** werden alle nicht leeren Zellen der Spalte A gezählt. Die Liste darf daher keine Lücken aufweisen! Die Zählvariable **i** stellt gleichzeitig die Zeilennumerierung der Tabelle dar. Vor der Zuweisung ihres Wertes an die globale Variable **`Endzeile`** muss sie um **1** verringert werden.
Diese Prozedur wird beim Starten des Programms ausgeführt.

Prozedur für das Change Ereignis des Registers: **`TabStrip1_Change`**:
In dieser Prozedur wird ermittelt, welcher Registerreiter gesetzt wurde – also gerade aktiv ist. Da unser Register-Steuerelement vier Register besitzt, geht die Indexnummer, die in der Eigenschaft Value gespeichert ist, von 0 bis 3. Auf Grund der Nummer wird über eine Select-Anweisung der Stringvariablen **`reg`** der zugehörige Text zugewiesen. Anschliessend wird in der **Funktion `Ausgaben`** die Summe der Beträge berechnet, die formatiert in einem Textfeld angezeigt wird. Dieses Textfeld ist über die Eigenschaft ***Locked*** für Eingabe gesperrt.

Funktion `Ausgaben`:
Als Parameter wird der Funktion der Name des aktiven Registers übergeben, der Rückgabewert ist vom Typ Currency, da es sich um ein Währungsfeld handelt. Die Liste wird von der zweiten Zeile ausgehend bis zur letzten durchlaufen. In der ersten Zeile befinden sich die Namen der Spalten. Die Preise der Artikel sind in Spalte B, die Mengen in Spalte C. Diese Werte werden in den Variablen **`preis`** und **`menge`** gemeinsam mit der Zeilennumerierung vermerkt.
Wenn der Registerreiter **Gesamt** aktiv ist, muss die gesamte Summe über die Liste der Artikel berechnet werden. Die Variable **`Laufsum`** speichert für jeden Artikel Preis mal Menge. Der numerische Wert der Zellen wird über die Eigenschaft ***Value*** zurückgegeben. Falls andere Register aktiv sind, muss in der Spalte Gruppe der Artikel immer daraufhin überprüft werden, ob er zu der entsprechenden Gruppe gehört.

Programmliste:

```
Private Blatt As Object
Private Endzeile As Integer

Private Sub UserForm_Initialize()
  Set Blatt = Sheets("Tabelle3")
  Blatt.Activate
  Listeermitteln
  TabStrip1.Value = 3
End Sub
```

```
Private Function Ausgaben(ByVal register As String) As Currency
  Dim i As Integer
  Dim gruppe As String, preis As String, menge As String
  Dim Laufsum As Currency
  For i = 2 To Endzeile
    preis = "B" & i
    menge = "C" & i
    If register = "Gesamt" Then
      Laufsum = Laufsum + Blatt.Range(preis).Value * _
                Blatt.Range(menge).Value
    Else
      gruppe = "E" & i
      If Blatt.Range(gruppe).Formula = register Then
        Laufsum = Laufsum+ Blatt.Range(preis).Value * _
                  Blatt.Range(menge).Value
      End If
    End If
  Next i
  Ausgaben = Laufsum
End Function

Sub Listeermitteln()
  Dim zelle As String
  zelle = "A1"
  Do Until Blatt.Range(zelle).Formula = ""
    i = i + 1
    zelle = "A" & CStr(i)
  Loop
  Endzeile = i - 1
End Sub

Private Sub TabStrip1_Change()
  Dim i As Byte, reg As String
  i = TabStrip1.Value
  Select Case i
    Case 0
      reg = "Hardware"
    Case 1
      reg = "Software"
    Case 2
      reg = "Zubehör"
    Case 3
      reg = "Gesamt"
  End Select
  TextBox1.Value = Format(Ausgaben(reg), "#,##0.00 sFr")
End Sub

Private Sub EndButton_Click()
  Unload Me
End Sub
```

Zugriff auf Datenbanken

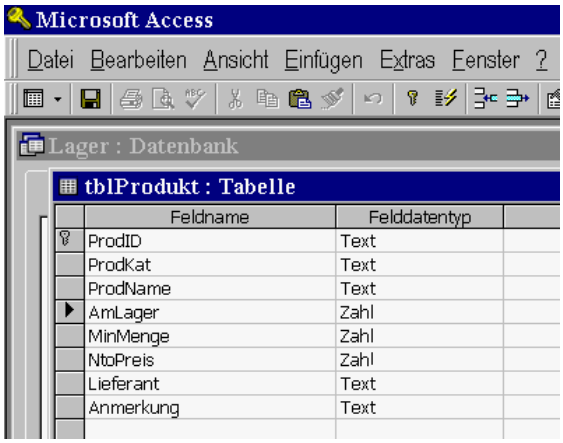
Aus EXCEL kann man auf die Datenbestände, die in Form einer Datenbank gespeichert sind, zugreifen z.B. auf **Access**, **dBase**, **FoxPro** usw. Eine Methode ohne die Datenbankapplikation zu starten, besteht mittels **D**ata **A**ccess **O**bject (**DAO**).

Beispiel: Computer-Hardware Handlung "Tech-Mut"

Die Firma Tech-Mut verkauft Computerkomponenten, aus denen sich der Kunde sein eigenes System zusammenstellen kann. Alle Angaben zu den Computerkomponenten sind in der ACCESS Datenbank ***lager.mdb*** gespeichert. Die Abteilung Marketing verwaltet das Produktesortiment und überwacht die Preisgestaltung, die Abteilung Lagerverwaltung ist für die Lagerhaltung und die Nachbestellung der Ware zuständig. Die Details für die Bearbeitung werden in diesem Beispiel nicht gezeigt. Wir zeigen im folgenden nur den Aufbau der Datenbank. Diese enthält nur eine Tabelle ***tblProdukt*** mit den angegebenen Feldern:

ACCESS-Datenbank: ***lager.mdb***
Tabelle: ***tblProdukt***

Datenbankentwurf:



Wenn der Kunde in den Laden kommt, kann er dem Verkäufer seine Wünsche angeben. Dieser erstellt ihm sofort eine Offerte. Das Layout für die Offerte ist in EXCEL erstellt und die Daten werden aus der Access Datenbank importiert. Die aktuellen Daten werden aus der Datenbank mit Hilfe des Datenbanktreibers **ODBC** (**O**pen **D**ata**B**ase **C**onnectivity) mit Verwendung von **DAO** direkt in die EXCEL-Tabelle eingetragen. Datenzugriffsobjekte (DAO) ermöglichen Ihnen, eine Programmiersprache zu verwenden, um auf Daten von lokalen und Remote-Datenbanken zuzugreifen.

Die EXCEL-Mappe ***lager.xls*** enthält die beiden Tabellenblätter ***Angebot*** und ***Produkt***. Im Tabellenblatt Angebot ist die Offerte der Firma Tech-Mut (Trivial Error Computer Hardware, Muttenz) dargestellt. Nach den Angaben des Kunden gibt der Verkäufer die Produktidentifikation (ProdID) gemäss der Produktliste ein. Falls der Kunde mehr als 1 Stück wünscht, muss die Menge eingegeben werden - andernfalls wird automatisch 1 Stück eingetragen. Wenn alle Produkte eingetragen sind, wird die **ANGEBOT**-Taste angeklickt. Dadurch wird mit Hilfe der Prozedur **`cmdAngebot_Click`** die abgebildete Tabelle automatisch erstellt. Die Prozedur ist durch Kommentare erläutert. Durch Anklicken der Taste **LÖSCHEN** (Prozedur **`cmdDel_Click`**) wird die Tabelle für eine neue Offerte vorbereitet.

Tech-Mut [Angebot] [Löschen]

Trivial Error Computer Hardware, Muttenz *1. April 1998*

Mein Computer

MWSt % 6.5

ProdID	Menge	Kategorie	Produktname	NtoPreis	MWSt	StückPreis	Preis
PII300	1	Computer	Intel Pentium II 300 MHz, 512 L2 Cashe	1635	106.28	1'741.30	1'741.30
GCDia3D	1	Grafikkarte	Diamond Stealth 3D 4000 4MB, 220 MHz	113	7.35	120.35	120.35
M32Dimm	2	Memory	32 MB SDIMM, 168pins, W/EEPROM	70	4.55	74.55	149.10
HD43IBM	1	Hard Disk	IBM IDE HD A34331, 4.3 GB, 9.5ms	320	20.80	340.80	340.80
CD32MI	1	CD-ROM	MITSUMI 32x speed IDE CD-ROM	125	8.13	133.15	133.15
Mo17B	1	Monitor	Belinea 17" Monitor, 95 kHz, 0.28 mm	999	64.94	1'063.95	1'063.95

1. Tragen Sie Produkt-Ident und Menge ein

2. Klicken Sie auf [Angebot]

Total **3'548.65**

Im Tabellenblatt ***Produkt*** befindet sich die folgende Übersicht. Durch Anklicken der Taste **AKTUALISIEREN** werden mit Hilfe der Prozedur **`cmdUpDate_Click`** die aktuellen Daten aus der Datenbank automatisch übernommen, ohne dass die Datenbank geöffnet wird.

Produkte am Lager

[Aktualisieren]

ProdID	ProdKat	ProdName	AmLager
P200	Computer	Intel Pentium 200 MHz, Pipeline Burst Cashe	6
P233	Computer	Intel Pentium 233 MHz, 512 L2 Cashe	6
PII266	Computer	Intel Pentium II 266 MHz, 512 L2 Cashe	6
PII300	Computer	Intel Pentium II 300 MHz, 512 L2 Cashe	6
PII400	Computer	Intel Pentium II 400 MHz, 512 L2 Cashe	6
GCDia3D	Grafikkarte	Diamond Stealth 3D 4000 4MB, 220 MHz	1
GCMMI4	Grafikkarte	Matrox Millenium 4MB RAM 250 MHz	3
GCMMY4	Grafikkarte	Matrox Mystique 4MB RAM 220 MHz	0
HD21Fu	Hard Disk	Fujitsu 2.1 GB HDD IDE	4
HD25S	Hard Disk	Seagate ST32520A, 2.5 GB ATA-3 IDE	0
HD32Q	Hard Disk	Quantum Fireball ST, 3.2 GB	3
HD43IBM	Hard Disk	IBM IDE HD A34331, 4.3 GB, 9.5ms	3
HD64IBM	Hard Disk	IBM IDE HD A36480, 6.4 GB, 9.5ms	1
CD32MI	CD-ROM	MITSUMI 32x speed IDE CD-ROM	2
M16Simm	Memory	16 MB SDRAM, 72pins, EDO no Parity	11
M32Simm	Memory	32 MB SDRAM, 72pins, EDO no Parity	11
M32Dimm	Memory	32 MB SDIMM, 168pins, W/EEPROM	14
M64Dimm	Memory	64 MB SDIMM, 168pins, W/EEPROM	5
M128Dimm	Memory	128 MB SDIMM, 168pins, W/EEPROM	0
Mo15B	Monitor	Belinea 15" Monitor, 69 kHz, 0.28 mm	8
Mo17B	Monitor	Belinea 17" Monitor, 95 kHz, 0.28 mm	5
Mo17E	Monitor	EIZO F56 17" Monitor, 95 kHz, 0.26 mm	1

Programmcode **DieseArbeitsmappe**:

```
' LAGER          Beispiel zur Anwendung von DAO (Data Access Object)
'                Version 1.0 vom 21.03.1998      Autor: Zdenko Jardas

' Zugriff zu Datenbanken über ODBC muss in Excel aktiviert werden:
'   - VBE → EXTRAS → VERWEISE
'   - Microsoft DAO 3.5 Object Library aktivieren und [OK] klicken

' In cmdAngebot_Click() und cmdUpdate_Click() muss das Verzeichnis,
' in dem sich die Datenbank befindet, angepasst werden:
'     sDbPath = "C:\User\Info_vba\"
```

Programmcode **Tabelle1**:

```
Private Sub cmdAngebot_Click()
          ' Diese Sub öffnet die Tabelle 'tblProdukt'
          ' in der Access-Datenbank "Lager.mdb",
          ' liest die ProdId aus der ersten Spalte in
          ' Excel-Tabelle "Angebot",
          ' sucht den Datensatz mit ProdId in der
          ' Datenbank-Tabelle und kopiert die Daten aus
          ' der Datenbank-Tabelle in die Excel-Tabelle
  Dim Db As Database, Rs As Recordset
  Dim sDbName As String, sDbPath As String
  Dim iZl As Integer
  sDbPath = "C:\User\Exel\VBA97\Unterlagen\Kapitel_9\"
  sDbName = "Lager.mdb"
          ' öffne die Datenbank (DB)
  Set Db = Workspaces(0).OpenDatabase(sDbPath & sDbName)
          ' öffne die Tabelle mit Produkten
  Set Rs = Db.OpenRecordset("tblProdukt")
  Rs.Index = "PrimaryKey"
  For iZl = 6 To 22
          ' Eintrag im 'Angebot' ist vorhanden
  If ActiveSheet.Cells(iZl, 1) <> "" Then
          ' Menge nicht eingetragen
    If ActiveSheet.Cells(iZl, 2) = "" Then
      ActiveSheet.Cells(iZl, 2) = 1       ' so setze die Menge 1
    End If
          ' Suche ProdId aus 'Angebot' in der DB
    Rs.Seek "=", ActiveSheet.Cells(iZl, 1)
    If Rs.NoMatch Then        ' ProdId ist nicht in DB
      ActiveSheet.Cells(iZl, 4) = "-Produkt nicht im Programm-"
    Else                      ' Kopiere Daten aus dem Datensatz
      ActiveSheet.Cells(iZl, 3) = Rs!ProdKat
      ActiveSheet.Cells(iZl, 4) = Rs!ProdName
      ActiveSheet.Cells(iZl, 5) = Rs!NtoPreis
    End If
  End If
  Next iZl
  Rs.Close
End Sub

Private Sub cmdDel_Click()
          ' Entfernt alle Einträge aus der Tabelle
  Range("A6:E22").ClearContents
End Sub
```

Programmcode **Tabelle2**:

```
Private Sub cmdUpdate_Click()
          ' Diese Sub öffnet die Tabelle "tblProdukt"
          ' in der Access-Datenbank "Lager.mdb",
          ' und kopiert alle Datensätze aus der Datenbank-
          ' Tabelle in die Excel-Tabelle "Produkt"
  Dim Db As Database, Rs As Recordset
  Dim sDbName As String, sDbPath As String
  Dim iZl As Integer
  sDbPath = "C:\User\Exel\VBA97\Unterlagen\Kapitel_9\"
  sDbName = "Lager.mdb"
  Set Db = Workspaces(0).OpenDatabase(sDbPath & sDbName)
  Set Rs = Db.OpenRecordset("tblProdukt")
          ' Entfernt alle Einträge aus der Tabelle
  Range("A4:D50").ClearContents
  iZl = 4
  Rs.MoveFirst
  Do While Not Rs.EOF
    With ActiveSheet
      .Cells(iZl, 1) = Rs!ProdId
      .Cells(iZl, 2) = Rs!ProdKat
      .Cells(iZl, 3) = Rs!ProdName
      .Cells(iZl, 4) = Rs!AmLager
    End With
    Rs.MoveNext
    iZl = iZl + 1
  Loop
  Rs.Close
Sub End
```

Fehlersuche in VBA - Programmen

Syntaktische Fehler "Schreibfehler"

Vor der Programmausführung:

→ **TESTEN** → **KOMPILIEREN VON VBA-PROJECT**

Fehler werden farblich gekennzeichnet.

Tip: kleine Programmabschnitte jeweils kontrollieren

Treten während der Ausführung Fehler auf, so beachten Sie die Meldung im Meldungsfenster und verwenden Sie die Schaltfläche **HILFE**.

Logische Fehler "Denkfehler"

VBE bietet Unterstützung in der Form von "***Debugging***".

Testen im **Unterbrechungsmodus**:

Das Programm wird nicht vollständig, sondern nur bis zu einem bestimmten Punkt ausgeführt und dann gestoppt. Das Anhalten wird durch Setzen von Haltepunkten erreicht.

Ausführen in **Einzelschritt** (Taste F8)

Dabei lassen sich folgende Fenster verwenden:

- ***Direktfenster***

  erlaubt das Ausführen von Prozeduren oder Funktionen (Angabe des entsprechenden Namens) und das Ansehen von Variablenwerten (entweder mit der Maus über den Variablennamen fahren oder ? und Variablennamen im Direktfenster angeben)

- ***Lokalfenster***

  zeigt automatisch alle deklarierten Variablen der aktuellen Funktion oder Prozedur und deren Werte im Unterbrechungsmodus an

- ***Überwachungsfenster***

  ähnlich wie das Lokalfenster, es zeigt aber nur die vom Benutzer angegebenen Variablen an (günstig bei vielen Variablen). Die gewünschten Variablen werden im Codefenster markiert und dann mit der Maus ins Überwachungsfenster gezogen.