2.1 Hinweise zum Umgang mit aufgezeichneten Makros

Wie bereits erwähnt, erlaubt der Makrorecorder von Excel das Erstellen von Makros ohne Programmierkenntnisse. Er zeichnet alle Anweisungen und Eingaben in Form von VBA-Befehlen auf, die Sie später bei Bedarf im VBA-Editor ansehen und bearbeiten können.

Die Begriffe Makros und Prozedur meinen im Grunde immer dasselbe, nämlich ein VBA-Programm das bestimmte Aufgaben ausführt. Allerdings wird zur besseren Unterscheidung der Begriff Makros häufig nur für, mit dem Makrorecorder aufgezeichnete, Makros verwendet.

> Leider zeichnet der Makrorecoder alle Arbeitsschritte und Eingaben, also auch misslungene Versuche auf. Überlegen Sie daher vor der Aufzeichnung, welche Arbeitsschritte in welcher Reihenfolge erforderlich sind und testen Sie die Schritte eventuell vorher. Auch ein kurzes Notieren der richtigen Reihenfolge kann hilfreich sein.

Bevor wir mit dem ersten Makro beginnen, einige Hinweise zu Makrorecorder. Dieser ist auch für fortgeschrittene Anwender in VBA-Kenntnissen nicht ganz überflüssig. Besonders dann, wenn einzelne VBA-Anweisungen nicht bekannt sind, ist es manchmal schneller, einzelne Befehle aufzuzeichnen und diese dann im VBA-Editor zu bearbeiten oder in die eigene Prozedur einzufügen.

Allerdings ist der aufgezeichnete VBA-Code meist wesentlich umständlicher als selbst geschriebene Prozeduren und enthält häufig auch überflüssige Anweisungen.

Zuletzt noch ein Tipp: Durch ein Makro ausgeführte Schritte können nicht mehr rückgängig gemacht werden. Speichern Sie also beim Testen von Makros die Arbeitsmappe, bevor Sie ein Makro ausführen. Dadurch vermeiden Sie, dass zwischenzeitlich vorgenommene Änderungen verlorengehen. Außerdem kann ein fehlerhaftes Makro Excel zum Absturz bringen.

2.2 Ein einfaches Makro aufzeichnen

Die Aufzeichnung starten

Als Beispiel wollen wir ein Makro aufzeichnen, das den Text „Hallo“ in eine zuvor markierte Zelle schreibt. So gehen Sie vor:

1 Markieren Sie eine Zelle, beispielsweise A1 und klicken Sie im Register *Entwicklertools*, Gruppe *Code*, auf *Makro aufzeichnen*.

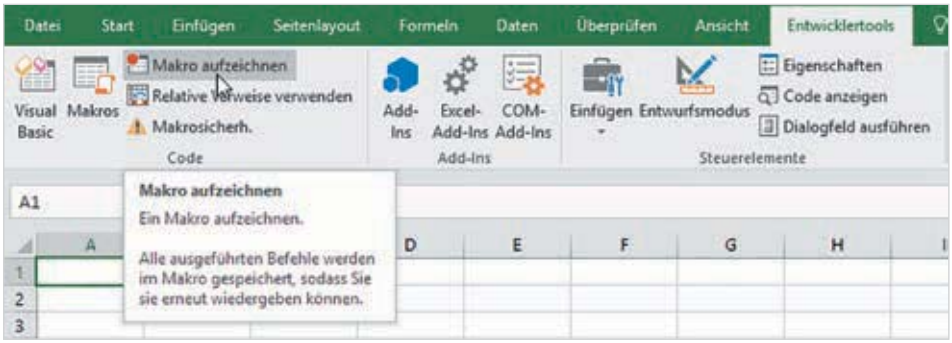


Bild 2.1 Klicken Sie auf Makro aufzeichnen

2 Das Dialogfenster *Makro aufzeichnen* wird geöffnet: Geben Sie einen Namen an, unter dem das Makro gespeichert und später aufgerufen werden soll.

Beachten Sie die Regeln für Makronamen: Ein Makroname muss mit einem Buchstaben beginnen, darf maximal 255 Zeichen lang sein und keine Leerzeichen und mit Ausnahme des Unterstrichs (_) auch keine Sonderzeichen enthalten, also auch keinen Bindestrich.

Bild 2.2 Makroaufzeichnung starten

3 **Tastenkombination zum Starten des Makros**
Falls Sie später das Makro über eine Tastenkombination starten möchten, so geben Sie die gewünschte Taste in Verbindung mit der Strg (Ctrl)-Taste an. Sie können ein Makro aber auch auf andere Weise starten oder dem Makro nachträglich eine Tastenkombination zuweisen.

Achtung: Die Tastenkombination unterscheidet zwischen Groß- und Kleinbuchstaben. Zudem sollten Sie eine Tastenkombination wählen, die nicht bereits anderweitig belegt ist, z. B. m, M, j, J.

4 **Speicherort festlegen**
Unter *Makro speichern in* wählen Sie den Speicherort des Makros. Hier haben Sie die Wahl zwischen der Persönlichen Makroarbeitsmappe und der aktuellen Ar-

beitsmappe. Die aktuelle Arbeitsmappe (*Diese Arbeitsmappe*) ist die Standardeinstellung und sollte im Normalfall beibehalten werden.

Diese Beschreibung erscheint später im VBA-Editor als Kommentar, siehe Kap. 3.

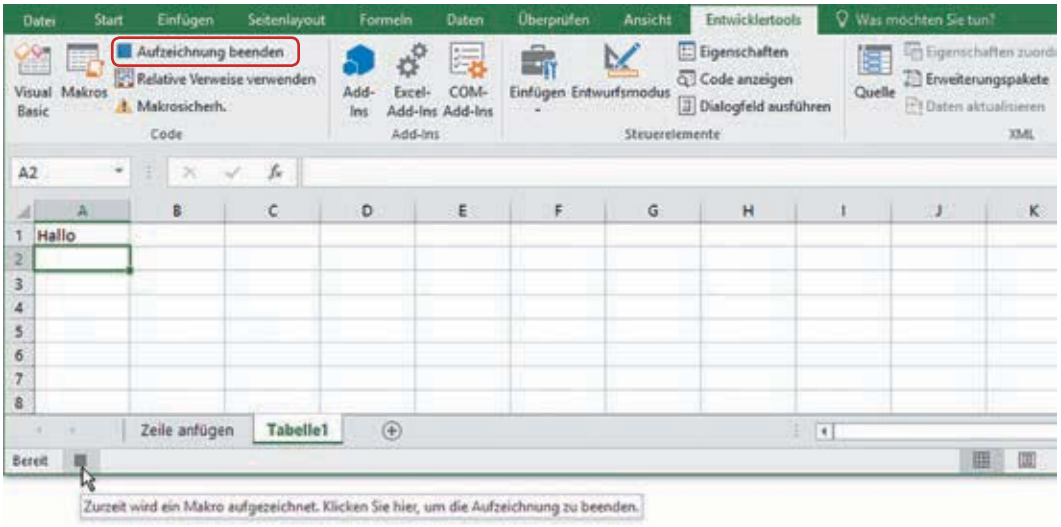
5 Im Feld *Beschreibung* können Sie optional eine kurze Beschreibung der Funktionsweise des Makros eingeben. Dies ist im Hinblick auf die spätere Nachvollziehbarkeit unbedingt zu empfehlen.

6 Mit der Schaltfläche *OK* starten Sie abschließend die Makroaufzeichnung.

7 Ab jetzt werden alle Ihre Befehle und Eingaben aufgezeichnet. Da in unserem Beispiel die Zelle A1 bereits vor der Makroaufzeichnung markiert wurde, tippen Sie das Wort „Hallo" in diese Zelle und drücken anschließend die Eingabe-Taste. Nun ist die Zelle A2 unterhalb markiert.

8 Zum Schluss müssen Sie die Makroaufzeichnung beenden: Klicken Sie dazu im Register *Entwicklertools*, Gruppe *Code*, auf *Aufzeichnung beenden*. Als Alternative können Sie auch die Schaltfläche in der Statusleiste verwenden, siehe Bild unten.

Bild 2.3 Aufzeichnung beenden



Makro ausführen

Um das soeben aufgezeichnete Makro zu testen, löschen Sie zuvor das Wort „Hallo" aus A1 und markieren dann eine beliebige andere Zelle des Tabellenblatts.

- Haben Sie dem Makro eine Tastenkombination zugewiesen, so verwenden Sie zum Starten des Makros diese Tastenkombination.
- Andernfalls klicken Sie im Register *Entwicklertools* ▸ *Code* auf die Schaltfläche *Makros.* Markieren Sie im Fenster *Makro* das Makro, das Sie ausführen möchten und klicken Sie auf die Schaltfläche *Ausführen*. Als Alternative öffnen Sie das Fenster *Makro* mit den Tasten Alt+F8.

Objekte, Methoden und Eigenschaften

Beim Umgang mit Excel-Objekten können Sie mit VBA diesen eine Methode zuweisen oder die Objekteigenschaften ändern.

Beispiel: Ein Excel Arbeitsblatt einfügen

Worksheets fassen alle Excel-Arbeitsblätter zusammen, wenn Sie beispielsweise ein weiteres Arbeitsblatt in die aktuelle Mappe einfügen möchten, geschieht dies mit der Anweisung *Worksheets.Add*.

Objekte und deren Methoden oder Eigenschaften werden mit einem Punkt (.) voneinander getrennt. Sobald Sie ein Objekt, im Bild unten *Worksheets*, gefolgt von einem Punkt, eingetippt haben, erscheint eine Liste mit allen, für das Objekt verfügbaren, Eigenschaften und Methoden, die Sie durch Markieren und Doppelklick oder Auswahl mit Pfeiltaste und anschließender Tab-Taste in die Anweisung übernehmen.

Bild 3.33 Eingabe von Methoden und Eigenschaften eines Objekts

Tipp: Die Liste der Eigenschaften und Methoden kann manchmal sehr umfangreich sein. In solchen Fällen gelangen Sie schneller zum Ziel, wenn Sie ein oder zwei Buchstaben eingeben.

Auch die Methode *Add* können Sie wieder durch die optionalen Parameter Einfügeposition, Anzahl und Typ näher spezifizieren. Wird hier nichts angegeben, so wird das neue Tabellenblatt links vom aktuellen Blatt eingefügt.

Bild 3.34 Parameter der Methode Add

Kommentare

Erläuterungen in Prozeduren erleichtern Ihnen und anderen die spätere Bearbeitung, deshalb sollten Sie ausgiebig von der Möglichkeit, Kommentare einzufügen, Gebrauch machen. Kommentare beginnen mit einem Hochkomma (Apostroph) ' und können als gesonderte Zeile oder rechts am Ende einer Anweisungszeile eingegeben werden. Sie erscheinen automatisch in grüner Schrift, siehe Optionen.

Kommentare werden bei der Ausführung der Prozedur ignoriert, daher werden auch keine Anführungszeichen für den Kommentartext benötigt. Kommentare mit Erklärungen sollten grundsätzlich zu Beginn einer Prozedur und an wichtigen Punkten eingefügt werden.

```
Public Sub Beispiel1()
'Ein erstes Beispiel zur Programmierung mit VBA
'Erstellt von Inge Baumeister 2018
'**************************************
MsgBox "Hallo!", vbOKCancel, "Erster Test"  'Meldung ausgeben

End Sub
```

Bild 3.35 Beispiele für Kommentare

Tipp: Anweisungen in Kommentare umwandeln und umgekehrt

In der Symbolleiste finden Sie das Symbol *Block auskommentieren*. Mit diesem können Sie eine oder mehrere markierte Anweisungszeilen schnell in Kommentarzeilen umwandeln. Dies ist äußerst nützlich, wenn Sie eine Anweisung aus der Prozedur vorübergehend entfernen möchten, ohne diese zu löschen. Handelt es sich um eine einzige Zeile, so genügt es, wenn sich hier der Cursor befindet.

Mit dem Symbol *Auskommentierung des Block aufheben* wandeln Sie die markierten Kommentarzeilen wieder in normale Anweisungen um.

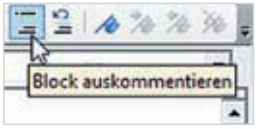


Bild 3.36 Anweisungszeilen in Kommentar umwandeln

Die VBA-Hilfe

Nützliche Dienste leistet die VBA-Hilfe. Sie rufen die allgemeine Hilfe mit einem Klick auf das Symbol der Symbolleiste oder über das Menü *?* und *Microsoft Visual Basic for Applications-Hilfe* auf. Hier ist vor allem die *Excel-VBA-Referenz* von Interesse. Neben einer allgemeinen Einführung in VBA in Excel finden Sie hier eine alphabetisch geordnete Übersicht über das Objektmodell von Excel sowie unter *Konzepte* verschiedene Themen und Tipps sowie komplette Lösungen zu häufigen Aufgabenstellungen.

Die Direkthilfe während der Eingabe nutzen

Direkthilfe: F1

Sehr zu empfehlen, nicht nur für Einsteiger und Gelegenheitsprogrammierer, ist die Direkthilfe mit der Taste F1. Mit dieser erhalten Sie eine vollständige Beschreibung aller Parameter einschließlich kleiner Beispiele. Als Beispiel auch hierzu wieder *MsgBox*:

Geben Sie *Msgbox* in eine Anweisungszeile ein, platzieren Sie den Cursor an beliebiger Stelle innerhalb des Wortes oder markieren Sie das Wort mit Doppelklick und betätigen Sie die Taste F1. Anschließend öffnet sich Ihr Standardbrowser mit einer detaillierten Beschreibung dieser Funktion. Hier finden Sie auch eine Übersicht über die Rückgabewerte der einzelnen Tasten, z. B. *OK*.

Achtung: Die Hilfe ist ausschließlich online verfügbar, d. h. nur wenn Sie mit dem Internet verbunden sind.

3.4 Variablen, Konstanten und Operatoren

In vielen Fällen müssen während der Ausführung einer Prozedur Zwischenergebnisse gespeichert werden oder werden mehrfach benötigt. In solchen Fällen benötigen Sie Variablen. Konstanten erfüllen im Prinzip denselben Zweck, mit dem einzigen Unterschied, dass Konstanten innerhalb der Prozedur ein fester Wert zugewiesen wird und dieser während der Laufzeit nicht geändert wird.

Variablen verwenden

Als Variablen werden in der Programmierung Platzhalter oder Behälter für Daten bezeichnet, denen erst während der Ausführung ein Wert zugewiesen wird. Vor der Verwendung sollte unbedingt jede Variable mit Name und Datentyp festgelegt werden, dies erfolgt mit der Anweisung *Dim*. Die Deklaration von Variablen sollte außerdem gleich am Beginn der Prozedur vor der ersten Anweisung erfolgen. Beispiele:

```
Dim Benutzername as String
Dim Alter as Integer
```

Variablennamen

Folgende Regeln sind bei Variablennamen zu beachten:

- Ein Variablenname muss mit einem Buchstaben beginnen und darf max. 255 Zeichen lang sein.
- Ein Variablenname muss innerhalb des Gültigkeitsbereichs eindeutig sein, darf also nicht mehrfach vorkommen.
- Ein Variablenname darf keine Leerzeichen, keine Sonder- oder Satzzeichen enthalten.
- Optional kann dem Variablennamen ein Präfix vorangestellt werden, das den Datentyp kennzeichnet, siehe Tabelle z. B. strName für die String-Variable Name.

Datentypen

Jede Variable sollte nicht nur mit Namen sondern auch mit ihrem Datentyp deklariert werden. VBA unterscheidet die folgenden Datentypen:

Typ	Bereich	Beispiel	Präfix
Byte	Ganze Zahlen von 0 bis 255	36	byt
Integer	Ganze Zahlen von -32.769 bis 32.768	12.345	int
Long	Ganze Zahlen von -2.147.483.648 bis 2.147.483.648	123.458	lng
Single	Dezimalzahlen mit 8 Stellen Genauigkeit	0,2234	sng
Double	Dezimalzahlen mit 16 Stellen Genauigkeit	0,1457003598112	dbl

Typ	Bereich	Beispiel	Präfix
Currency	Festkommazahl mit 15 Stellen vor und 8 Stellen hinter dem Komma	12,90	cur
Date	Datum und Uhrzeit	15.03.2016	dat
String	beliebige Zeichenfolge, alphanumerisch	"Feldweg 7a"	str
Variant	beliebige Zeichenfolge, Zext oder Zahlen	"Frieda" oder 999	var
Boolean	Wahr oder Falsch, True / False	True	bln
Object	alle Objektreferenzen	Worksheet	obj

Bei Deklaration von Variablen ist die Angabe des Datentyps nicht zwingend erforderlich. Allerdings sind solche Variablen automatisch vom Typ *Variant*. Das bedeutet, der Variablen kann jeder beliebige Inhalt, egal ob Text oder Zahl zugewiesen werden. Dies wiederum kann während der Ausführung zu Laufzeitfehlern führen, wenn z. B. statt des eingegebenen Textes eine Zahl zur Berechnung erwartet wird. **Beispiel**: *Dim GebDatum As Date* weist der Variablen den Typ *Datum* zu, mit der Anweisung *Dim GebDatum* dagegen ist die Variable vom Typ *Variant*.

Variablendeklaration erzwingen

Die Deklaration von Variablen ist in der Standardeinstellung des VBA-Editors nicht zwingend erforderlich, sollte aber unbedingt vorgenommen werden. Mit der Anweisung *Option Explicit* zu Beginn eines Moduls im sogenannten Deklarationsbereich, erzwingen Sie eine Deklaration aller verwendeten Variablen. Dadurch werden auch Tippfehler bei der Eingabe von Variablennamen in einer Prozedur schnell erkannt, da Sie bei Verwendung einer nicht deklarierten Variablen eine Fehlermeldung erhalten. In den Optionen (*Extras* ▸ *Optionen*) können Sie festlegen, dass diese Anweisung automatisch zu Beginn jedes neuen Moduls eingefügt wird.

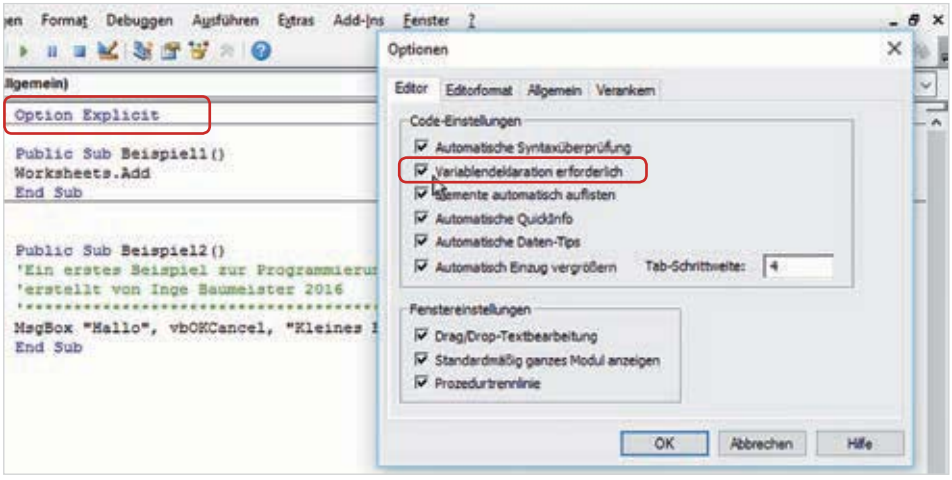


Bild 3.37 Variablendeklaration erzwingen

Auf diese Weise braucht diese Anweisung nicht jedes Mal manuell in ein neues Modul eingegeben werden.

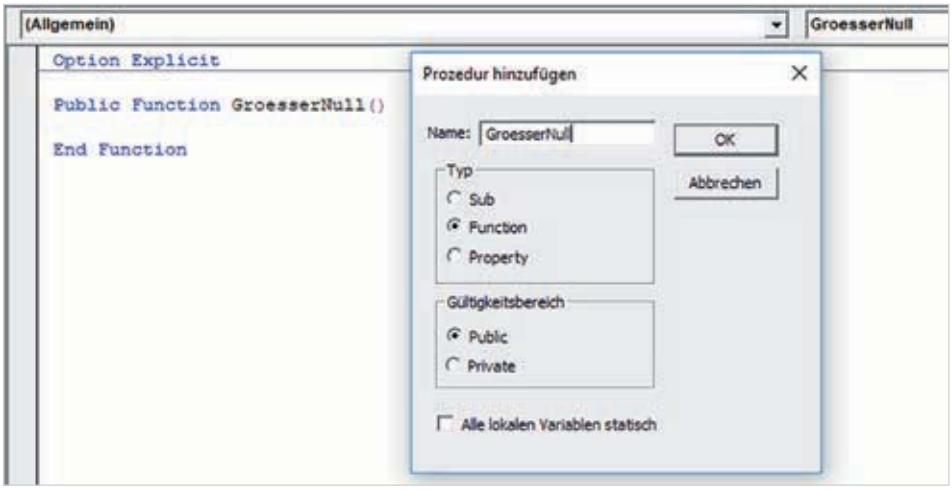


Bild 3.62 Funktion einfügen

Die nachfolgende Funktion *GroesserNull* prüft, ob der Parameter *Zahl* größer ist als 0 und liefert *True* oder *False* als Rückgabewert (Boolean).

```
Public Function GroesserNull(Zahl As Long) As Boolean
    If Zahl > 0 Then
        GroesserNull = True
    Else
        GroesserNull = False
    End If
End Function
```

Bild 3.63 Die Funktion prüft, ob eine Zahl größer als 0 ist

Funktionsaufruf und Übergabe der Parameter

Zum Aufruf der Funktion geben Sie den Funktionsnamen und dahinter in Klammern den oder die erforderlichen Parameter an. Wie bei jeder Funktion erscheinen während der Eingabe Parameter und Datentyp als Infotext. Im Bild unten wird das Ergebnis der Funktion der Variablen *Ergebnis* zugewiesen.

```
Ergebnis=groessernull(
          GroesserNull(Zahl As Long) As Boolean
```

Bild 3.64 Parameter und Datentyp

```
Sub Funktionstest()
'Testet die Funktion GroesserNull
    Dim EingabeZahl As Long
    Dim Ergebnis As Boolean

    EingabeZahl = InputBox("Bitte geben Sie eine Zahl ein:")
    Ergebnis = GroesserNull(EingabeZahl)
    MsgBox "Das Ergebnis lautet: " & Ergebnis
End Sub
```

Bild 3.65 Die vollständige Prozedur

Achten Sie bei Funktionen nicht nur beim Namen sondern auch bei der Benennung der Parameter auf aussagekräftige Bezeichnungen.

Bild 3.66 Mehrere Parameter definieren

Mehrere Parameter übergeben

Wenn mehrere Parameter erforderlich sind, müssen diese einzeln, mit Komma getrennt, mit Name und Datentyp festgelegt werden, z. B.:

```
Public Function ParameterBeispiel(strNachname As String, intAlter As Integer) As String
```

Funktion ausführen/testen

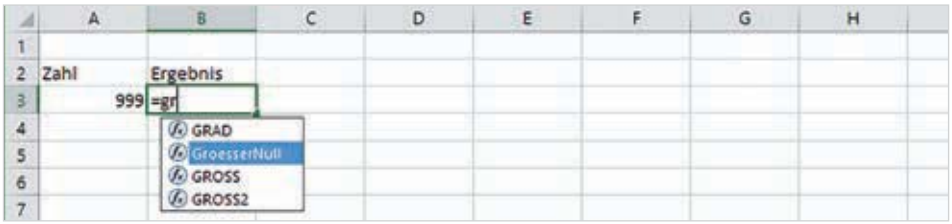
Zum Testen von Funktionen und Routinen eignet sich das *Ausführen*-Symbol nicht. Sie sollten daher zu Testzwecken eine kleine Prozedur schreiben, mit der Sie die Funktion aufrufen, siehe oben.

Ausnahme: Funktionen die als Private deklariert wurden, stehen im Arbeitsblatt nicht zur Verfügung!

Funktionen im Arbeitsblatt verwenden

Mit VBA erstellte Funktionen sind auch im Arbeitsblatt verfügbar und erscheinen, sobald Sie in einer Zelle nach dem Gleichheitszeichen die ersten Zeichen der Funktion eingegeben haben, siehe Bild 3.67. Wie Sie Funktionen für den Einsatz in Tabellen mit einer Beschreibung benutzerfreundlicher gestalten und als Add-In speichern, wird detailliert in Kapitel 4.7 beschrieben.

Bild 3.67 Funktion im Arbeitsblatt verwenden



Prozeduren als Routine

Prozeduren liefern im Gegensatz zu Funktionen keinen Rückgabewert. Sie dienen in erster Linie zum Ausführen kleiner Aufgaben, z. B. Einfügen von Arbeitsblättern oder Formatieren von Zellbereichen. Auch an Prozeduren können Parameter übergeben werden, diese werden genau wie bei Funktionen im Prozedurkopf mit Name und Datentyp festgelegt. Mehrere Parameter werden mit Komma getrennt, wobei für jeden Parameter der Datentyp festgelegt werden muss, siehe oben.

Als Beispiel eine kleine Prozedur mit dem Namen *BlaetterEinfügen* zum Einfügen von Arbeitsblättern, die gewünschte Anzahl wird mit dem Parameter *intAnzahlBlaetter* übergeben.

Bild 3.68 Beispiel Prozedur als Routine

```
Public Sub BlaetterEinfügen(intAnzahlBlaetter As Integer)
'Fügt die gewünschte Anzahl Arbeitsblätter
'links vom aktuellen Blatt ein
    Worksheets.Add Count:=intAnzahlBlaetter
    MsgBox intAnzahlBlaetter & " Tabellenblätter erfolgreich eingefügt!"
End Sub
```

Prozedur ausführen

Prozeduren, denen als Routinen Parameter übergeben werden, können Sie nur testen, indem Sie zum Aufrufen eine kleine Prozedur schreiben. Es genügt, wenn Sie in die Anweisungszeile einfach nur den Prozedurnamen eingeben, im Gegensatz zu Funktionen folgen dahinter nach einem Leerzeichen die Parameter ohne Klammern.

```
Sub TestBlattEinfügen()
'Test der Routine BlaetterEinfügen
    Dim intAnzahl As Integer
    intAnzahl = InputBox("Wieviele Blätter möchten Sie einfügen?")

    BlaetterEinfügen intAnzahl
End Sub
```

Bild 3.69 Prozeduraufruf und Übergabe der Parameter

3.9 Mit Datenfeldern arbeiten

Datenfelder oder Arrays, wie sie in der Programmierung allgemein bezeichnet werden, stellen eine Sonderform der Variablen dar. Sie können gleich mehrere Variablen aufnehmen und die einzelnen Elemente des Arrays werden über ihren Index adressiert. Deklarationsanweisung und Gültigkeitsbereich von Arrays entspricht normalen Variablen mit einem Unterschied: Die Anzahl der Elemente muss mit einem Indexwert für Ober- und Untergrenze angegeben werden. Ist keine Untergrenze definiert, so erhält das erste Element den Indexwert 0 (Nullbasierte Indizierung). VBA unterscheidet außerdem zwischen ein- und mehrdimensionalen Datenfeldern.

Beachten Sie eine wichtige Bedingung: Alle Elemente eines Datenfeldes müssen vom gleichen Typ sein, ausgenommen beim Datentyp Variant.

Eindimensionale Datenfelder

Deklaration und Wertezuweisung

Die Deklaration von Datenfeldern erfolgt mit der Anweisung `Dim` zusammen mit Name und Datentyp, wobei dem Namen in Klammern die Anzahl der aufzunehmenden Elemente hinzugefügt wird.

```
Dim arrFeld(AnzahlElemente) As Datentyp
```

Als Beispiel soll ein Datenfeld deklariert werden, das später 5 Namen aufnehmen soll. Folgende Anweisungen sind möglich, wobei mit der ersten Anweisung das erste Element den Indexwert 0 erhält und mit der zweiten Anweisung den Index 1.

```
Dim arrName(4) As string
Dim arrName(1 To 5) As string
```

Die Wertezuweisung erfolgt über den Index, wie im Bild unten. Elemente, denen kein expliziter Wert zugewiesen wurde, bleiben leer (Nullwerte).

Bild 3.70 Wertezuweisung an ein eindimensionales Datenfeld

```
Sub eindimensionalesDatenfeld()
'Namen einem Datenfeld zuweisen

Dim strName(1 To 5) As String

strName(1) = "Otto"
strName(2) = "Frieda"
strName(3) = "Theodor"
strName(4) = "Emil"
strName(5) = "Sabine"

End Sub
```

Beachten Sie bei der Adressierung über den Index

Ein Index beginnt in VBA standardmäßig mit 0. Wird keine Untergrenze angegeben sondern nur die Anzahl der Elemente, z. B. *Dim strName(4)*, dann erhält das erste Element den Index 0. Damit das erste Element eines Datenfeldes den Index 1 erhält, muss 1 als Untergrenze angegeben werden, z. B. *Dim StrName(1 To 5)*.

Diese Möglichkeit wird in diesem Buch nicht verwendet!

Tipp: Wenn Sie am Anfang eines Moduls die Anweisung *Option Base 1* eingeben, dann beginnt innerhalb dieses Moduls automatisch jeder Index mit 1. Die explizite Angabe von 1 als Untergrenze ist dann nicht mehr erforderlich.

Bild 3.71 Untergrenze 1 im Modul festlegen

Die Verwendung von LBound und UBound

Bild 3.72 Werte über Index auslesen

Bild 3.73 Werte mit LBound und UBound auslesen

Zum Auslesen der Werte des Datenfeldes, z. B. um sie in die Zellen eines Arbeitsblattes zu schreiben oder einfach nur über *Debug.Print* im Direktfenster auszugeben, muss jedes Element über seinen Index adressiert werden. Es bietet sich eine *For...Next*-Schleife an, in der einfachsten Form wie in Bild 3.72 abgebildet.

```
Sub EindimensionalesDatenfeld()
'Namen einem Datenfeld zuweisen

Dim strName(1 To 5) As String
Dim z As Integer

strName(1) = "Otto"
strName(2) = "Frieda"
strName(3) = "Theodor"
strName(4) = "Emil"
strName(5) = "Sabine"

'Werte im Direktfenster ausgeben
For z = 1 To 5
    Debug.Print strName(z)
Next

End Sub
```

```
Sub EindimensionalesDatenfeld2()
'Namen einem Datenfeld zuweisen

Dim strName(1 To 5) As String
Dim z As Integer

strName(1) = "Otto"
strName(2) = "Frieda"
strName(3) = "Theodor"
strName(4) = "Emil"
strName(5) = "Sabine"

'Werte im Direktfenster ausgeben
For z = LBound(strName) To UBound(strName)
    Debug.Print strName(z)
Next

End Sub
```

In Zelle B7 eine Zahl schreiben
```
Worksheets("Tabelle1").Cells(7, 2).value = 777
```

Ohne Angabe des Arbeitsblatts bezieht sich *Cells* auf das aktuelle Blatt. Beispiel: In A5 des aktuellen Arbeitsblatts das Wort „Test" schreiben
```
Cells(5,1).Value = "Test"
```

Die Angabe eines Zellbereichs erfordert zusätzlich *Range*. Beispiel: A1 bis C5 markieren
```
Range(Cells(1, 1), Cells(5, 3)).Select
```

Beispiel: Zellen nacheinander mittels Schleife adressieren

Das folgende Beispiel fügt am Ende ein neues Tabellenblatt ein und füllt hier die Zellen A1 bis A50 mit den Zahlen von 1 bis 50 aus. Die Zählervariable kann daher gleichzeitig als Zellinhalt zugewiesen werden.

Bild 4.19 Zellen mittels Schleife ausfüllen

```
Sub ZahlenAusfuellen()
'Fügt am Ende ein neues Blatt ein
'und füllt A1:A50 mit den Zahlen von 1 bis 50 aus

    Dim objNeuesBlatt As Worksheet
    Dim z As Long                          'Zählervariable
    Set objNeuesBlatt = Worksheets.Add(after:=Worksheets(Worksheets.Count))

    objNeuesBlatt.Select
    For z = 1 To 50
        Cells(z, 1).Value = z
    Next
End Sub
```

Der Vorteil der Eigenschaft *Cells*: Zeile und/oder Spalte können mit Variablen vom Typ Long angegeben werden.

Arbeiten mit der aktiven Zelle bzw. dem markierten Zellbereich

Die aktive bzw. markierte Zelle

Activate und Select entspricht dem Markieren von Zellen

Wenn zuvor mit der Methode *Activate* oder *Select* eine Zelle ausgewählt wurde, stellt diese die aktive Zelle dar und kann über *ActiveCell* angesprochen werden. *ActiveCell* gibt ein Range-Objekt zurück, daher sind auch für *ActiveCell* alle Eigenschaften und Methoden des Range-Objekts verfügbar, z. B. einen Wert zuweisen oder Inhalte löschen.

```
ActiveCell.Value = 777
ActiveCell.ClearContent
```

Wurde ein Zellbereich markiert, so kann trotzdem nur eine Zelle die *ActiveCell* sein, in diesem Fall immer die Zelle in der linken oberen Ecke des Zellbereichs.

Beachten Sie: *ActiveCell* kann sich nur im aktuell ausgewählten Arbeitsblatt befinden. Eventuell muss daher zuvor mit *Worksheets(Tabelle).Select* das Tabellenblatt ausge-

wählt werden, bevor Sie mit *Range* oder *Cells* eine Zelle oder einen Zellbereich auswählen.

```
Sub AktuelleZelle()
' Zelle C3 im Blatt Tabelle1 mit
' roter Hintergrundfarbe formatieren

    Worksheets("Tabelle1").Select
    Range("C3").Activate
    ActiveCell.Interior.Color = vbRed

End Sub
```

Bild 4.20 ActiveCell muss sich im aktuellen Blatt befinden

Der gesamte aktive Zellbereich

Wenn ein Zellbereich markiert wurde und die Anweisung den gesamten Zellbereich einbeziehen soll, dann verwenden Sie *Selection* statt *ActiveCell*. Die folgende Anweisung wählt das Arbeitsblatt Tabelle1 aus, markiert hier den Bereich A1 bis C10 und löscht anschließend alle Inhalte und Formate des markierten Bereichs.

```
Sub InhalteLoeschen()
' Betreich A1:C10 im Blatt Tabelle1 markieren
' und Inhalte löschen

    Worksheets("Tabelle1").Select
    Range("A1:C10").Activate
    Selection.Clear

End Sub
```

Bild 4.21 Inhalte eines Bereichs löschen

Markierung mit der Offset Methode verschieben

Die *Offset* Methode bezieht sich auf die aktive Zelle (*ActiveCell*) und verschiebt die Markierung um die angegebene Zeilen- und Spaltenzahl. Die Syntax:

```
ActiveCell.Offset(Zeilen, Spalten)
```

Beispiel: Die folgenden Anweisungen markieren im aktuellen Tabellenblatt zuerst die Zelle B10 und verschieben anschließend die Markierung in derselben Spalte um 5 Zeilen nach unten. Dadurch wird die Zelle B15 markiert.

```
Range("B10").Select
ActiveCell.Offset(5, 0).Select
```

Auch negative Angaben sind möglich, dann wird die Markierung nach links bzw. oben verschoben. Die folgende Anweisung verschiebt die Markierung um 1 Zeile nach oben und 3 Spalten nach links.

```
ActiveCell.Offset(-1, -3).Select
```

Position der aktuellen Zelle ermitteln

Um die aktuelle Position, beispielsweise der markierten Zelle oder das aktive Arbeitsblatt zu ermitteln, können die folgenden Anweisungen verwendet werden:

Beschreibung	Anweisung
Die Adresse der aktuell markierten/ aktiven Zelle des aktuellen Arbeitsblattes in der Form A1	`ActiveCell.Adress`
Die Zeilennummer der aktiven Zelle	`ActiveCell.Row`
Die Spaltennummer der aktiven Zelle	`ActiveCell.Column`
Name der aktuellen Tabelle	`ActiveSheet.Name` `ActiveCell.Parent.Name`
Name der Arbeitsmappe	`ActiveSheet.Parent.Name`

Hinweis: Die Eigenschaft *Parent* steht für das jeweils übergeordnete Objekt. Bei einem Zellbereich ist dies das Arbeitsblatt, in einem Arbeitsblatt die Arbeitsmappe.

Beispiele

- Die Adresse der aktuell markierten Zelle in der Schreibweise A1 ermitteln und in einem Meldungsfenster ausgeben. Wenn ein Zellbereich markiert ist, dann liefert die Prozedur die Adresse der Zelle in der linken oberen Ecke des Bereichs.

Bild 4.22 Adresse der markierten Zelle in der Schreibweise A1

```
Sub AdresseZelle()
'Adresse der aktuell markierten Zelle
'in der Schreinweise $A$1

    Dim adresse As String
    adresse = ActiveCell.Address
    MsgBox "Zelle " & adresse & " ist markiert!"
End Sub
```

Bild 4.23 Das Ergebnis

- Die Adresse der markierten Zelle in der Schreibweise Zeile, Spalte ermitteln. Dies entspricht *Cells(*Zeile, Spalte*)*.

Bild 4.24 Zeile und Spalte ermitteln

Die dazugehörige Prozedur:

```
Sub AdresseZeileSpalte()
'Adresse der aktuell markierten Zelle
'in der Schreibweise Zeile, Spalte

    Dim Zeile As Long
    Dim Spalte As Long
    Zeile = ActiveCell.Row
    Spalte = ActiveCell.Column
    MsgBox "Zelle in Zeile " & Zeile & " und Spalte " & Spalte & " ist markiert!"
End Sub
```

Bild 4.25 Zeile und Spalte der markierten Zelle

- Die unten abgebildete Prozedur ermittelt den Namen des aktuellen Tabellenblatts. Statt *ActiveCell.Parent.Name* könnte auch *ActiveSheet.Name* verwendet werden, da sich die aktive Zelle in jedem Fall im aktuellen Blatt befindet.

```
Sub NameTabellenblatt()
'Tabellenblatt anzeigen in dem sich die mrkierte Zelle befindet

    Dim BlattName As String
    BlattName = ActiveCell.Parent.Name
    MsgBox "Der Name des aktuellen Arbeitsblatts: " & BlattName
End Sub
```

Bild 4.26 Name des aktuellen Tabellenblatts

Umfang eines Zellbereichs ermitteln

Auch in VBA ist Excel in der Lage, zusammenhängende Tabellenbereiche zu erkennen. Häufig geht es ja darum, einen Zellbereich auszuwerten, dessen genauer Umfang nicht bekannt oder veränderbar ist. Hierzu verwenden Sie die Eigenschaft *CurrentRegion*, die ebenfalls ein *Range* Objekt zurückgibt. So markiert beispielsweise die folgende Anweisung den Zellbereich, der die aktuell markierte Zelle umgibt.

```
ActiveCell.CurrentRegion.Select
```

Als Beispiel das Markieren eines Zellbereichs im Tabellenblatt Namensliste. Da A1 einen festen Bezugspunkt für die angegebene Tabelle (siehe Bild 4.28) bildet, wird diese Zelle zuvor ausgewählt.

```
Sub BereichMarkieren()
'Zellbereich ab A1 markieren
    Worksheets("Namensliste").Select
    ActiveSheet.Range("A1").Select
    ActiveCell.CurrentRegion.Select
End Sub
```

Bild 4.27 Zellbereich um die aktive Zelle markieren

	A	B	C	D	E	F	G	H	I
1	*Nachname*	*Vorname*	*Geburtsdatum*						
2	Müller	Sabine	01.06.1981						
3	Brösel	Werner	15.01.1972						
4	Kabelschacht	Alfred	03.08.1985						
5	Fröhlich	Frieda	11.10.1965						
6	Waldbach	Horst	24.05.1970						
7									
8									

Bild 4.28 Das Ergebnis im Tabellenblatt

Beispiel Zellbereich formatieren

Die Eigenschaft *CurrentRegion* kann beispielsweise zum Formatieren eines Zellbereichs eingesetzt werden, dessen Umfang nicht bekannt ist. Hier ein Beispiel:

Die Zellen der unten abgebildeten Tabelle sollen Schriftgröße 8 und rote Schrift erhalten. Unabhängig von der Anzahl der Zeilen und Spalten beginnt die Tabelle A1, daher wird zuerst diese Zelle aktiviert.

Bild 4.29 Beispiel

	A	B	C	D	E	F	G	H
1	Nachname	Vorname	Geburtsdatum					
2	Müller	Sabine	01.06.1981					
3	Brösel	Werner	15.01.1972					
4	Kabelschacht	Alfred	03.08.1985					
5	Fröhlich	Frieda	11.10.1965					
6	Waldbach	Horst	24.05.1970					
7								
8								

Bild 4.30 Die dazugehörige Prozedur

```
Sub BereichFormatieren()
'Zellbereich in Schriftgröße 8 und roter Schrift formatieren

    Worksheets("Namensliste").Select
    ActiveSheet.Range("A1").Select
    ActiveCell.CurrentRegion.Select
    With Selection.Font
        .Size = 8
        .Color = vbRed
    End With
End Sub
```

Anzahl der Zeilen und Spalten ermitteln

Mit *CurrentRegion* lässt sich auch die Anzahl der Zeilen und Spalten eines Zellbereichs ermitteln, um so beispielsweise nur Zeilen oder Spalten zu durchlaufen. Verwenden Sie dazu die folgenden Anweisungen

```
ActiveCell.CurrentRegion.Rows.Count
ActiveCell.CurrentRegion.Columns.Count
```

Die folgende Prozedur ermittelt für die Tabelle im Blatt Namensliste (siehe Bild 4.29) die Anzahl der Zeilen und Spalten und gibt diese in einem Meldungsfenster aus.

Bild 4.31 Anzahl der Zeilen und Spalten eines Zellbereichs

```
Sub AnzahlZeilenSpalten()
'Anzahl der Zeilen und Spalten im Blatt Namensliste ermitteln

    Dim Zeilen As Long
    Dim Spalten As Long

    Worksheets("Namensliste").Select
    ActiveSheet.Range("A1").Select
    Zeilen = ActiveCell.CurrentRegion.Rows.Count
    Spalten = ActiveCell.CurrentRegion.Columns.Count

    MsgBox "Die Tabelle umfasst " & Zeilen & " Zeilen und " & Spalten & " Spalten"
End Sub
```

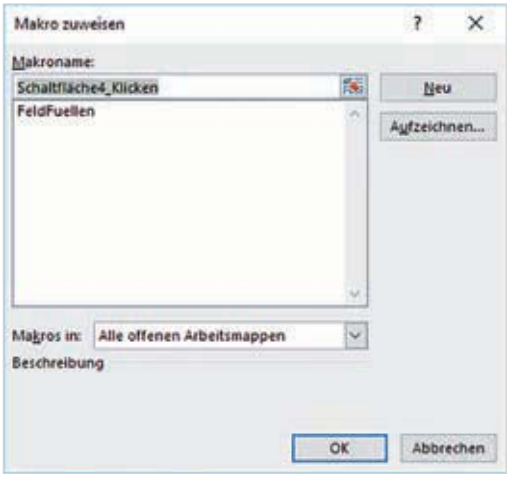
5.3 Beispiele für Formularsteuerelemente

Makro einer Befehlsschaltfläche zuweisen

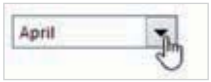
- Unmittelbar nach dem Einfügen einer Befehlsschaltfläche öffnet sich automatisch das Fenster *Makro zuweisen*. Markieren Sie das gewünschte Makro und klicken Sie zum Übernehmen auf *OK*.
- Falls das Makro noch nicht vorhanden ist, können Sie mit einem Klick auf die Schaltfläche *Aufzeichnen...* die Aufzeichnung starten. In diesem Fall sollten Sie dem Makro einen aussagekräftigeren Namen geben, da das Makro sonst den Namen *Schaltfläche_Klicken* erhält, siehe Bild unten.

Bei Bedarf können Sie über die rechte Maustaste und den Befehl *Makro zuweisen* dieses Fenster jederzeit wieder öffnen und der Schaltfläche ein anderes Makro zuweisen. Über die Schaltfläche *Code anzeigen* (Register *Entwicklertools*) wird der VBA-Editor mit dem dazugehörigen Programmcode geöffnet.

Bild 5.5 Makro zuweisen



Kombinationsfeld und Listenfeld zur Auswahl nutzen



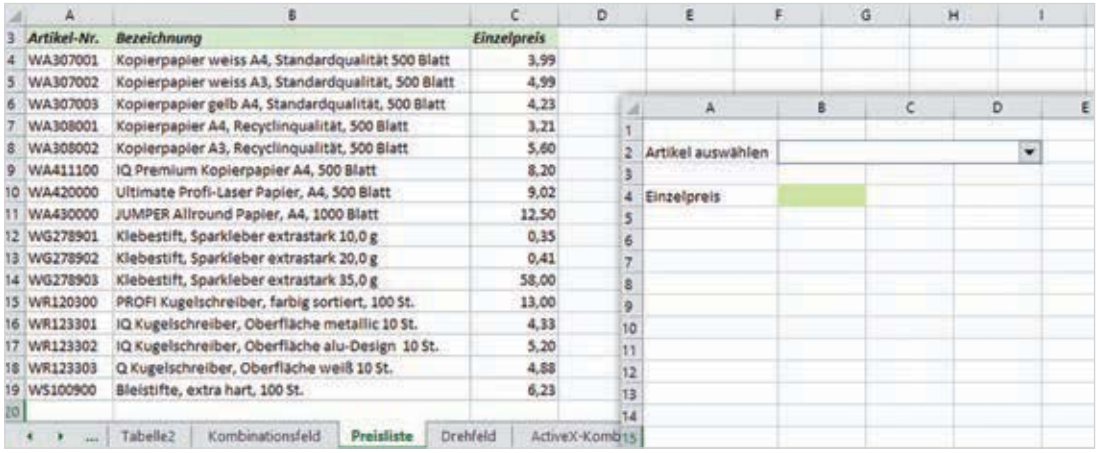
Kombinations- und Listenfelder zeigen eine Auswahlliste an und geben den markierten Wert in eine zuvor festgelegte Zelle aus. Allerdings liefern beide Felder nur den Zeilenindex, d. h. der wievielte Wert der Liste wurde ausgewählt, Sie benötigen daher in den meisten Fällen zusätzlich eine Funktion, z. B. INDEX, um den dazugehörigen Zellinhalt zu ermitteln.

Beispiel: Sie möchten über ein Kombinationsfeld einen Artikel auswählen und der Preis dieses Artikels soll anschließend im Arbeitsblatt angezeigt werden.

1. Im ersten Schritt fügen Sie in das Tabellenblatt ein Kombinationsfeld zusammen mit den erforderlichen Beschriftungen ein (Bild unten).

2 Dann benötigen Sie noch eine Liste mit Werten, die im Kombinationsfeld angezeigt werden sollen, in unserem Beispiel die Preisliste. Diese kann sich entweder im selben oder einem anderen Arbeitsblatt befinden. In diesem Beispiel befindet sich die Liste im Blatt *Preisliste*.

Bild 5.6 Kombinationsfeld und Datenherkunft



Artikel-Nr.	Bezeichnung	Einzelpreis
WA307001	Kopierpapier weiss A4, Standardqualität 500 Blatt	3,99
WA307002	Kopierpapier weiss A3, Standardqualität, 500 Blatt	4,99
WA307003	Kopierpapier gelb A4, Standardqualität, 500 Blatt	4,23
WA308001	Kopierpapier A4, Recyclinqualität, 500 Blatt	3,21
WA308002	Kopierpapier A3, Recyclinqualität, 500 Blatt	5,60
WA411100	IQ Premium Kopierpapier A4, 500 Blatt	8,20
WA420000	Ultimate Profi-Laser Papier, A4, 500 Blatt	9,02
WA430000	JUMPER Allround Papier, A4, 1000 Blatt	12,50
WG278901	Klebestift, Sparkleber extrastark 10,0 g	0,35
WG278902	Klebestift, Sparkleber extrastark 20,0 g	0,41
WG278903	Klebestift, Sparkleber extrastark 35,0 g	58,00
WR120300	PROFI Kugelschreiber, farbig sortiert, 100 St.	13,00
WR123301	IQ Kugelschreiber, Oberfläche metallic 10 St.	4,33
WR123302	IQ Kugelschreiber, Oberfläche alu-Design 10 St.	5,20
WR123303	Q Kugelschreiber, Oberfläche weiß 10 St.	4,88
WS100900	Bleistifte, extra hart, 100 St.	6,23

3 Im nächsten Schritt klicken Sie mit der rechten Maustaste auf das Kombinationsfeld und auf *Steuerelement formatieren...*. Im Register *Steuerung* legen Sie nun die Steuerungsparameter fest

Eingabebereich

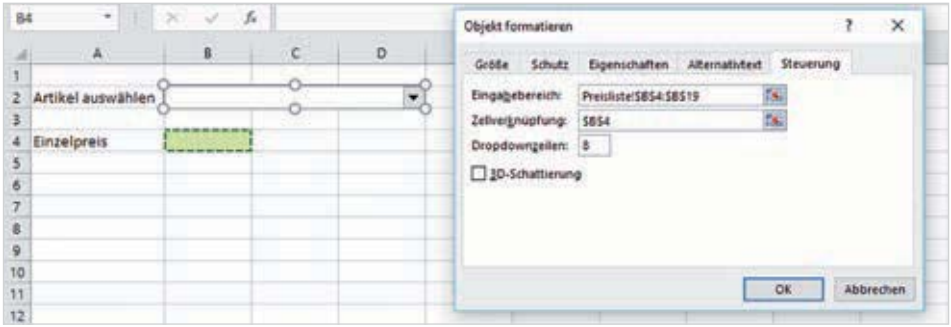
Hinweis: Befindet sich der Eingabebereich in einem anderen Tabellenblatt, so ist in älteren Excel-Versionen hierfür ein Bereichsname erforderlich!

Woher stammen die Werte des Kombinationsfeldes? Klicken Sie in das Feld *Eingabebereich* und markieren Sie anschließend im Tabellenblatt den gewünschten Bereich. Beachten Sie, dass das Formularsteuerelement Kombinationsfeld immer nur eine einzige Spalte anzeigen kann, Sie können daher in unserem Beispiel entweder die Spalte *Artikelnummer* oder *Bezeichnung* angeben. In unserem Fall verwenden wir die Bezeichnung, also den Bereich B4:B19.

Zellverknüpfung

Im Feld *Zellverknüpfung* geben Sie an, welche Zelle den ausgewählten Wert anzeigen soll, in unserem Beispiel C4. Unter *Dropdownzeilen* können Sie ggf. noch angeben, wie viele Zeilen das geöffnete Kombinationsfeld gleichzeitig anzeigen soll.

Bild 5.7 Steuerungsparameter festlegen



Vor dem Testen müssen Sie die Markierung mit einem Klick an eine beliebige Stelle des Arbeitsblattes aufheben. Klicken Sie dann auf den Dropdown-Pfeil des Kombinationsfeldes und auf eine Bezeichnung, sofort erscheint in der verknüpften Zelle der Zeilenindex des ausgewählten Wertes.

Bild 5.8 Das fertige Kombinationsfeld

Jetzt benötigen Sie noch in der Zelle rechts daneben (C4) die Funktion INDEX, um hier den Einzelpreis zu ermitteln, diese muss lauten:

=INDEX(Preisliste!B4:C19;B4;2)

Der Zeilenindex in B4 dient eigentlich nur als Zwischenergebnis und kann durch Formatieren mit entsprechender Schriftfarbe unsichtbar gemacht werden.

> Bei einem Listenfeld unterscheidet sich die Vorgehensweise nicht, so dass Sie dieses Beispiel auch mit einem Listenfeld testen können.

Kontrollkästchen

Fügen Sie das Formularsteuerelement Kontrollkästchen in ein Arbeitsblatt ein und ändern Sie die Beschriftung. Öffnen Sie dann das Fenster *Steuerelement formatieren* und legen Sie die folgenden Steuerungsparameter fest:

- **Zellverknüpfung:** In welche Zelle soll der Wert WAHR oder FALSCH ausgegeben werden? Im Bild unten in Zelle A2.
- **Wert:** Hier legen Sie die Standardeinstellung des Kontrollkästchens fest.

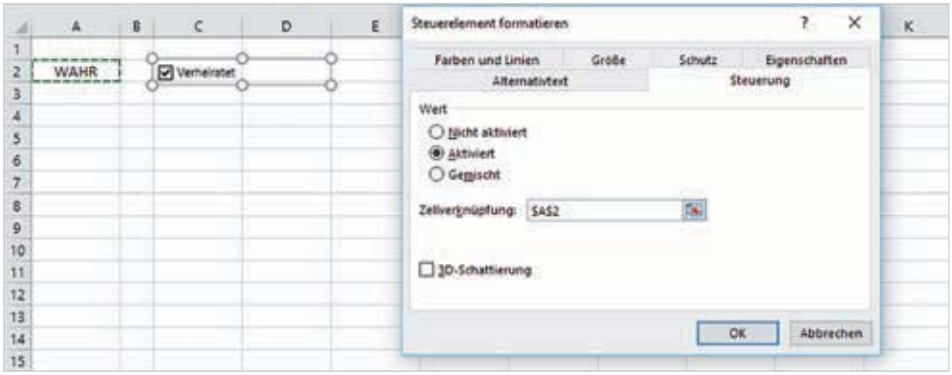


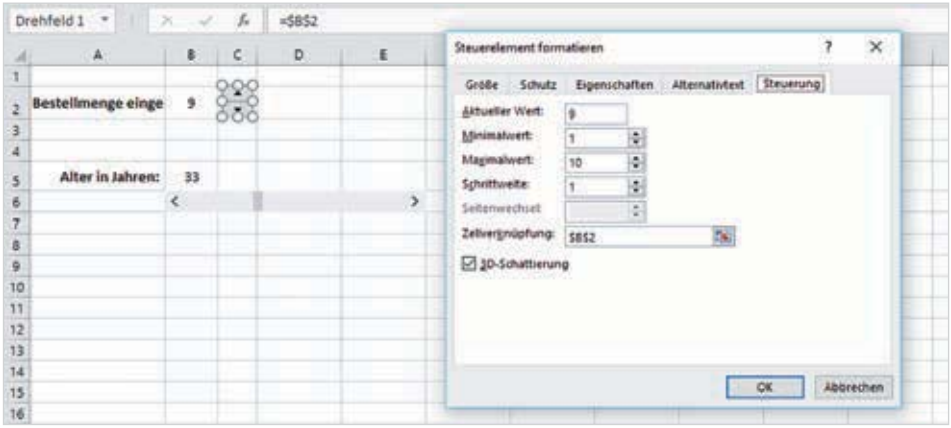
Bild 5.9 Steuerungsparameter Kontrollkästchen

Die Werte WAHR/FALSCH können anschließend in weiteren Formeln ausgewertet werden, z. B. in Verbindung mit der Funktion WENN.

Drehfeld und Bildlaufleiste

Die Steuerelemente Drehfeld und Bildlaufleiste eignen sich zur Eingabe bzw. Auswahl von Zahlen. Als Beispiel fügen Sie ein Drehfeld in das Arbeitsblatt ein und öffnen das Fenster *Steuerelement formatieren*. Als Steuerungsparameter werden benötigt: Minimal- und Maximalwert, sowie die Schrittweite um die mit jedem Mausklick hochgezählt wird. Unter *Aktueller Wert* geben Sie den Ausgangswert des Steuerelements an. Im Feld *Zellverknüpfung* geben Sie wieder an, welche Zelle den ausgewählten Wert erhalten soll.

Bild 5.10 Steuerungsparameter Drehfeld



Ähnlich verhält sich auch eine Bildlaufleiste, diese kann waagrecht oder senkrecht eingefügt werden. Sie benötigt dieselben Steuerungsparameter und unterscheidet sich vom Drehfeld nur dadurch, dass hier die Auswahl eines Wertes durch Verschieben mit gedrückter Maustaste erfolgt. Oder klicken Sie auf die kleinen Pfeile rechts und links bzw. oben und unten. In beiden Fällen kann in die verknüpfte Zelle auch einfach eine Zahl eingetippt werden.

Weitere Steuerelementeigenschaften

Namen zuweisen

Wenn Sie sich später z. B. in einem Makro auf ein Steuerelement beziehen möchten, dann sollten Sie ihm einen Namen zuweisen, da Namen wie *Drehfeld5* auf Dauer wenig aussagekräftig sind. Dazu markieren Sie das Steuerelement, klicken dann in das Namenfeld der Bearbeitungsleiste und geben hier den Namen ein. Schließen Sie durch Drücken der Eingabe-Taste ab.

Über den Namensmanager im Register *Formeln* können Sie ebenfalls Steuerelementen einen Namen geben. Markieren Sie zuvor das Steuerelement, klicken Sie auf *Namensmanager* und anschließend auf die Schaltfläche *Neu.....*.

6.5 Eigenschaften von Steuerelementen

Die Eigenschaften der Steuerelemente lassen sich mit VBA Code leicht verändern. **Ausnahme**: Die Namen der Objekte müssen fest vergeben werden.

Beschriftungsfeld (Label) verwenden

Wenn der Text von Beschriftungsfeldern per Programmanweisung geändert werden soll, erfolgt das entweder im Codefenster der Eingabemaske mit

```
Me.Label1.Caption = "Willkommen zum Workshop"
```

oder aus einem (anderen) Modul heraus mit

```
Eingabemaske.Label1.Caption = "Willkommen  um Workshop"
```

Anzeigetext

Über die Eigenschaft *Caption* wird dem Beschriftungsfeld der Anzeigetext zugewiesen. Die Textübergabe erfolgt auch ohne deren explizite Angabe.

Bild 6.48 Beschriftungsfeld als Hinweis

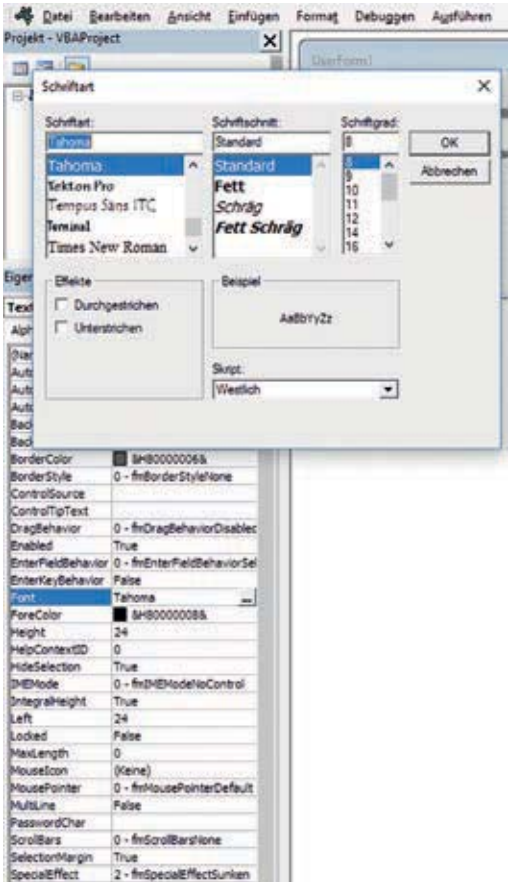


Bild 6.49 Schrifteigenschaften

Schrift

Schriftart, -größe, –schnitt und -farbe können im Eigenschaftenfenster unter *Font* und *ForeColor* eingestellt/angepasst werden.

Zeilenumbruch

Zeilenumbruch: Shift+Enter

Einen Zeilenumbruch für mehrzeiligen Text erzeugen Sie mit der Tastenkombination Shift+Enter.

Textfeld (TextBox) verwenden

Textfelder können ähnlich wie Beschriftungsfelder zur Anzeige von Informationen verwendet werden. Meist sind es allerdings Zellinhalte aus Tabellen, seltener Hinweise allgemeiner Art.

```
Eingabemaske.TextBox1.Value = "Wir werden sofort beginnen"
```

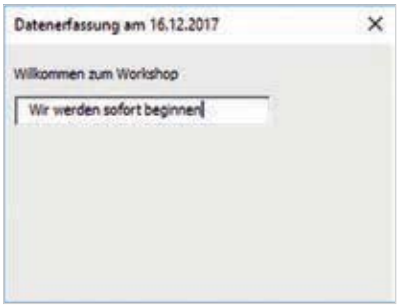
In unserem Beispielprojekt werden Textfelder zur Eingabe und Ausgabe von Tabellendaten verwendet.

Mehrzeilige Textfelder

Textfelder sind auf einzeilige Eingaben voreingestellt. Für mehrzeilige Eingaben muss die Eigenschaft *MultiLine = True* gesetzt werden und bei der Texteingabe jedes Mal die Tastenkombination Shift + Enter zum Zeilenumbruch verwendet werden. Um diesen Umstand abzustellen, kann die Eigenschaft *EnterKeyBehavior = True* gesetzt werden.

Zeilenumbruch: Shift+Enter

Achtung: Das Textfeld kann nun nicht mehr über Enter verlassen werden, sondern nur über die Tabulator-Taste oder durch Mausklick ins nächste Eingabefeld.



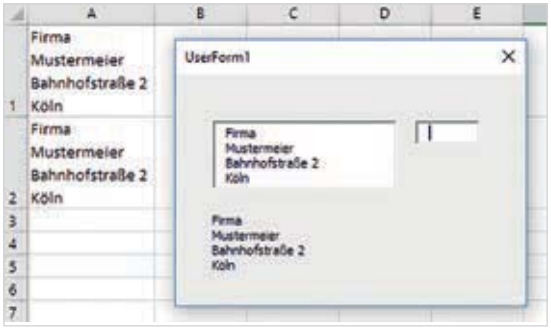


Bild 6.50 Textfeld für Hinweise und/oder Eingaben

Bild 6.51 Mehrzeiliges Textfeld

Die Eigenschaften *Text* bzw. *Value* geben die Zeilenaufteilung wieder, d.h. beim Ablegen in einer Tabellenzelle entsteht (erwartungsgemäß) ein mehrzeiliger Inhalt. Der Nutzen dieser mehrzeiligen Eingabeoption erschließt sich nicht jedem Anwender.

Optionsfeld (OptionButton) verwenden

Optionsfelder stehen in Formularen für bestimmte Auswahlmöglichkeiten, von denen jeweils nur eine einzige Variante aktiviert (*True*) werden kann. Mehrfachauswahl in derselben Gruppe ist nicht möglich (vgl. Kontrollkästchen). Optionsfelder haben ihr eigenes Beschriftungsfeld (*Caption*), das entweder rechts (Standard) oder links vom Kreissymbol steht (*Alignment*).

Jedes Optionsfeld kann entweder Wahr (*True*) oder Falsch (*False*) sein. Somit reicht es aus, eines der Felder als gesetzt vorzugeben (s. Abb. linke Gruppe).

```
Eingabemaske.OptionButton1.Value = True
```

Werden mehrere getrennte Auswahlbereiche angeboten, müssen die Optionsfelder gruppiert werden. Dazu gibt es zwei Möglichkeiten:

- Gemeinsame Gruppennamen vergeben (*GroupName*)
- oder sie in Rahmen-Elemente (Frame) einbetten.

Tipp: Um zu verhindern, dass eine Vorauswahl beim Erstellen der Eingabemaske getroffen wird, ohne dass diese bewusst gesetzt wurde, bietet es sich an, eine zusätzliche Option einzurichten, die als verdecktes Optionsfeld (*Visible = False*) programmiert werden kann (s. Bild 6.53, rechte Gruppe).

Bild 6.52 Verdecktes Optionsfeld (Formularansicht)

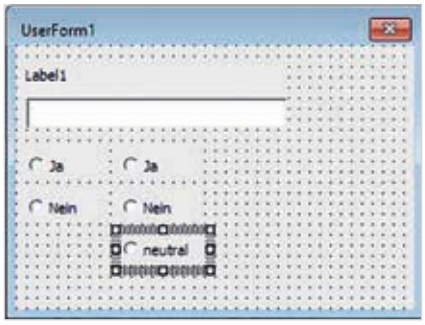


Bild 6.53 Verdecktes Optionsfeld (Codefenster)

Rahmen (Frame) verwenden

Rahmen eignen sich als grafische Elemente zur optischen Gestaltung des Formulars. Sie haben eine eingelassene Überschrift (*Caption*) und können beliebige Steuerelemente aufnehmen. Häufig werden sie zur Gruppierung von Optionsfeldern eingesetzt, um deren Gruppenzugehörigkeit (ohne Verwendung der Eigenschaft *GroupName*) herzustellen (siehe oben) und sie auch sichtbar abzugrenzen.

Bild 6.54 Gruppe3 über ein Rahmen-Element gruppiert (mit verdecktem Optionsfeld)

Kontrollkästchen (CheckBox) verwenden

Sollen mehrere Angaben zu einem Themenbereich auswählbar sein (Mehrfachauswahl), kommen die Kontrollkästchen zum Einsatz. Hin und wieder werden sie auch zur Darstellung eines logischen Zustandes (wahr/falsch) verwendet. Gruppierungen über die *GroupName*-Eigenschaft sind hier nicht wirksam.

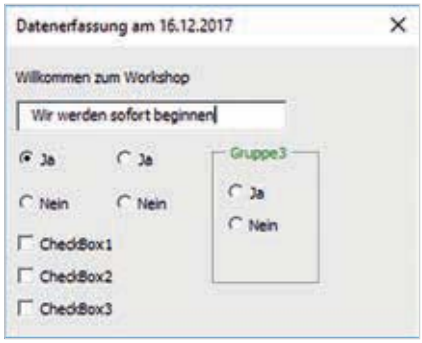


Bild 6.55 Kontrollkästchen für Mehrfachauswahl

Kombinationsfeld (ComboBox) verwenden

Ein Kombinationsfeld erscheint ähnlich einem Textfeld auf dem Formular. Neben der Möglichkeit, einen eigenen Eintrag vorzunehmen (Grundeinstellung), gibt das Kombinationsfeld in einer Auswahlliste mögliche bzw. erlaubte Einträge vor. Das Symbol mit einem kleinen schwarzen Pfeil am rechten Rand öffnet die Auswahlliste (Dropdownliste).

Bild 6.56 Kombinationsfeld mit freier Eingabemöglichkeit

Nur Vorgabewerte?

Neben der kombinierten Eingabe von vorgegebenen und eigenen Einträgen erlaubt das Kombinationsfeld über die Eigenschaft *Style* die Beschränkung der Eingabe nur auf die Vorgabewerte. Sie können zwischen Kombination= *DropDownCombo* und nur Liste (*DropDownList*) wählen.

Statische Zuweisung der Inhalte

Der Inhalt der Auswahlliste kann durch VBA Code zeilenweise vorgegeben bzw. erweitert werden. Eine andere Möglichkeit besteht darin, einen Zellbereich einer Tabelle zu definieren, der die gewünschten Einträge enthält (*RowSource*), siehe Bild unten.

SpecialEffect	2 - fmSpecialEffectSunken
Style	2 - fmStyleDropDownList
TabIndex	0

Bild 6.57 Einschränken der Eingaben auf Dropdownliste (Vorgaben)

MousePointer	0 - fmMousePointerDefault
RowSource	Hilfstabelle!A2:A7
SelectionMargin	True

Bild 6.58 Verweis auf Datenquelle für Vorgabewerte im Eigenschaftenfeld

```
Sub Auswahlliste_zuweisen()

    'Zellbereich fest vorgeben
    Kombinationsfeld_Demo.ComboBox1.RowSource = "Hilfstabelle!A2:A7"

End Sub
```

Bild 6.59 Verweis auf Datenquelle für Vorgabewerte durch Code

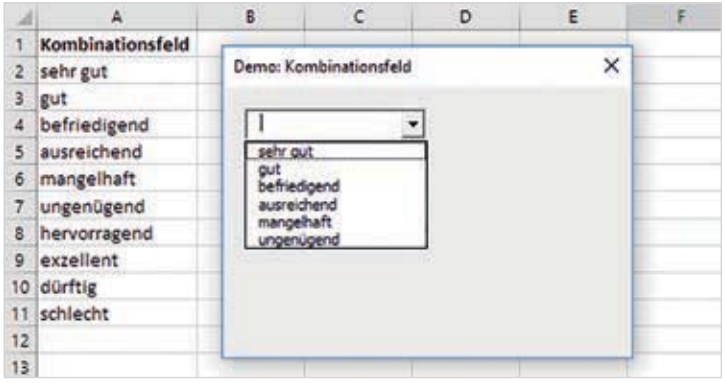


Bild 6.60 Auswahlliste aus Hilfstabelle (statisch)

Linienelemente und Einrahmungen

Für gestalterische Elemente, wie horizontale oder vertikale Trennlinien, gibt es keine speziellen Steuerelemente. Wenn Sie dennoch nicht auf derartige Effekte verzichten wollen, müssen Sie in die Trickkiste greifen. Es eignen sich zum Beispiel:

- **Rahmenelement**
  Dass sich ein Rahmenelement (*Frame*) zum Gruppieren von Optionsfeldern eignet, haben wir bereits erwähnt. In seiner Grundeinstellung macht er bereits eine rechteckige Einteilung optisch deutlich. Wenn auf die Überschrift (*Caption*) verzichtet wird, erscheint der Rahmen geschlossen.

- **Anzeigeelement/Image**
  Auch ein Anzeigeelement (*Image*) lässt sich zum Einrahmen von Objekten verwenden, wenn man es nicht mit einem Bild füllt.

Bei beiden Varianten ist es ratsam, zuerst das Rahmen- oder das Anzeigeelement im Formular zu positionieren und danach alle thematisch verknüpften Objekte hinein zu schieben. Die eingefügten Objekte liegen dann über dem zuerst gezeichneten Element.

Das Beispiel unten zeigt die Verwendung von Anzeige- und Rahmenelementen mit unterschiedlichen Eigenschaften. Mit ihrer Hilfe lassen sich Einrahmungen und Linienelemente in das Formular einbauen. Linien sind nichts Anderes als dünne (*Height* oder *Width*) Rahmen, die mit einer Hintergrundfarbe (*BackColor*) ausgefüllt sind und optional eine Umrahmung haben. Die Eigenschaft *SpecialEffect* stellt Ihnen noch weitere kreative Gestaltungsmöglichkeiten zur Verfügung.

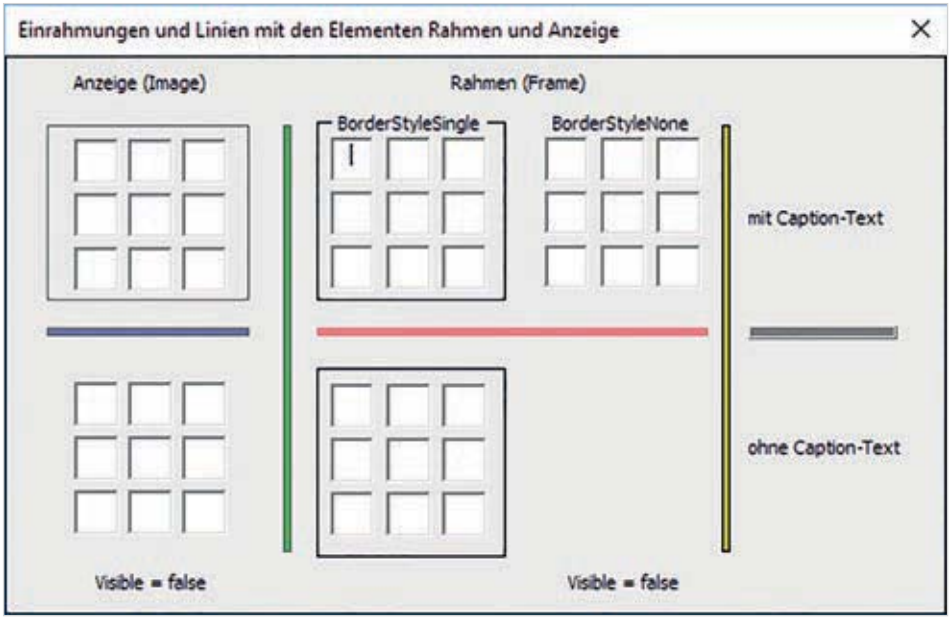


Bild 6.130 Anzeige (Image) und Rahmen (Frame) als Gestaltungselemente

Die einzelnen Unterschiede, auch unter Verwendung der Eigenschaft unsichtbar

Anzeige	Eigenschaften	Bemerkung
Anzeige oben links	*BorderStyle: BorderStyleSingle* *Visible: True*	
Anzeige unten links	*BorderStyle: BorderStyleSingle* *Visible: False*	es verschwindet nur der Rahmen
Rahmen oben Mitte	*BorderStyle: BorderStyleSingle* *Visible = True* mit Caption-Text	
Rahmen unten Mitte	*BorderStyle: BorderStyleSingle* *Visible = True* ohne Caption-Text	
Rahmen oben rechts	*BorderStyle: BorderStyleNone* mit Caption-Text	
Rahmen unten rechts	*BorderStyle: BorderStyleSingle* *Visible = False;*	Rahmen und alle darin befindlichen Objekte verschwinden
Trennlinien aus Anzeigeelementen	*BackColor: blau, grün* (Palette) *BorderStyle: BorderStyleNone)* *SpecialEffect: Flat*	
Trennlinien aus Rahmenelementen	*BackColor: rot, gelb, grau* ohne/mit Rahmen *SpecialEffect:Flat, Bump*	

7 Formulare als Dialogelemente einsetzen

In diesem Kapitel lernen Sie...

- Dialogelemente individuell gestalten
- Programmabläufe über Ereignisse (Initialize, Click, Change …) steuern
- Prozeduren mit Tastenkombinationen starten
- Anzeigewerte formatieren
- Umwandlungsfunktionen einsetzen

Das sollten Sie bereits wissen

- Prozedurrumpf erzeugen
- Variablentypen und Gültigkeitsbereiche
- Eigenschaften von Steuerelementen
- Aufrufvarianten von UserForms
- Codefenster der Userforms
- Abfragen und Verwendung von Schleifen
- MessageBox verwenden

Mit geringem Programmieraufwand lassen sich mithilfe der Formulare individuelle und vor allem unterschiedliche Dialogelemente bauen. Die nachfolgend beschriebenen Einzelschritte führen nach und nach zu unserem umfassenden Beispiel-Projekt.

Beginnen Sie am besten mit einer neuen Arbeitsmappe, die sie gleich zu Beginn als Excel-Arbeitsmappe mit Makros (.xlsm) in ihrem Arbeitsverzeichnis speichern. Diese Maßnahme erleichtert Ihnen das Testen von Prozeduren, wenn Sie vor deren Ausführung kurz das Speichersymbol oder die Kombination Strg + S betätigen. Für die Übungen treffen wir uns „Backstage" in der Entwicklungsumgebung (Alt + F11). Ich warte dort auf Sie.

7.1 Die individuelle MessageBox

In den vorausgegangenen Kapiteln zur Einführung in die VBA Programmierung haben Sie bereits einfache Dialogelemente kennengelernt: Die Messagebox (*MsgBox*) und die *InputBox*. Beide Elemente sind in ihrem Erscheinungsbild schlicht und sachlich gestaltet. Hin und wieder möchte man aber ein etwas ansprechenderes Dialogfeld als besonderen Hinweis oder als Blickfang einsetzen.

Beginnen wir mit einem Willkommensgruß zum Workshop. Sie benötigen dazu nur ein kleines Formular mit Beschriftungsfeld und ein geeignetes Bild für die Hintergrundgestaltung. Die notwendigen Einstellungen nehmen Sie im Eigenschaftenfenster der Steuerelemente vor.

- Fügen Sie im Projektfenster ein Formular (*UserForm*) ein.
- Geben Sie dem Formular einen Namen (*Name*) z. B. *Hinweis_Willkommen* .
- Für das Hintergrundbild wird die Pfadangabe benötigt (*Picture*).
- Passen Sie die Größe des Formulars ihren Vorstellungen entsprechend an (Anfasser).
- Die Farbe des Hintergrunds (*BackColor*) kann aus der Palette ausgewählt werden.
- Geben Sie dem Formular einen Titel (*Caption*) z. B. BILDNER Verlag.
- Passen Sie ein Beschriftungsfeld (*Label*) in das Formular ein.
- Fügen Sie dem Beschriftungsfeld einen Anzeigetext (*Caption*) hinzu z. B. „Willkommen zum Workshop".
- Starten Sie die Formularanzeige (Taste F5) und führen Sie gegebenenfalls Änderungen durch.

Bild 7.1 Die individuelle Messagebox

8.2 Die Eingabemaske für Tabellendaten aktivieren

Nachdem die Eingabemaske für das Auge fertig gestellt ist, muss sie noch für das Datenhandling vorbereitet werden. In den folgenden Übungen werden Sie mit den wichtigsten Grundlagen zum Erstellen von VBA-Befehlen kontinuierlich vertrauter werden. Immer das Ziel vor Augen, die Daten aus der Eingabemaske zu übernehmen, sie in einer Tabelle abzulegen und diese dann anschließend wieder in der Maske anzuzeigen.

„Viele Wege führen nach Rom" aber auch zu Lösungen mittels Visual Basic for Applications (VBA). Diese Programmiersprache stellt nahezu unendlich viele Möglichkeiten bereit, mit Daten und Elementen (Objekten) umzugehen. Ziel dieses Workshop-Abschnitts ist es, verständliche Wege aufzuzeigen, die leicht nachvollziehbar sind und sich nicht in hochkomplizierten Optimierungen verlieren. Programmieren soll Spaß machen, trotz der Fleißarbeit und Konzentration, die dazu abverlangt werden.

Vorbereitungen zum Start der Eingabemaske

Anmeldung_02.xlsm

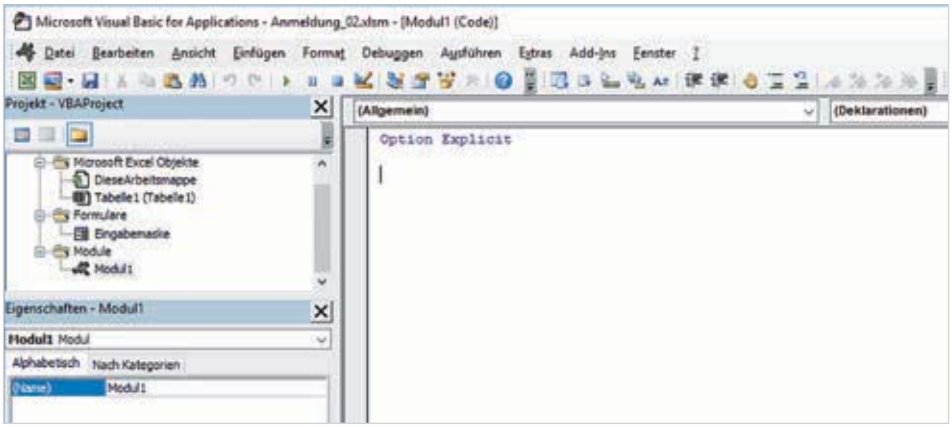
Starten wir gleich mit der gespeicherten Arbeitsmappe *Anmeldung_01.xlsm*. Noch bevor wir Änderungen vornehmen, speichern wir sie erneut, aber unter dem Namen *Anmeldung_02.xlsm*.

Der Grund: Wenn wir anschließend Prozeduren schreiben und ihre Wirkung testen, sollte vorweg immer erst ein Speichern erfolgen, um zu verhindern, dass beispielsweise bei Endlosschleifen der Rechner neu gestartet werden muss und wertvoller Programmcode verloren ist.

Modul einfügen

Statt Programmcode zusammen mit dem Formular (behind form) zu speichern, fügen wir ein Modul ein *Einfügen* ▸ *Modul*. Module sind nötig, um Programmcode themenbezogen – im Sinnzusammenhang – ablegen zu können. Sie übernehmen dieselbe Aufgabe wie Ordner oder Container. Ein Modul wird ähnlich wie ein Formularblatt (*UserForm*) im VBA-Projekt-Fenster eingefügt und erhält automatisch den Namen *Modul1*.

Bild 8.14 Modul einfügen



Dieses Modul soll Befehle umfassen, die die Eingabemaske vorbereiten. Das bedeutet, dass gewisse Grundeinstellungen beim Aufrufen der Eingabemaske vorgenommen werden sollen. Es bietet sich daher an, dem Modul einen passenden Namen zu geben wie beispielsweise *Formularinhalte*. Nach dem Einfügen des Moduls öffnet sich das Programmierfeld im Code-Fenster und wir sehen einen Hinweis in der ersten Zeile *Option Explicit* (siehe Bild 8.14). Diese Grundeinstellung verpflichtet uns, alle Variablen, die wir im Programmcode verwenden wollen, vorab zu deklarieren. Aber dazu kommen wir noch.

Hinweis: Alternativ ließen sich die Grundeinstellungen auch bei der Initialisierung der Eingabemaske vornehmen. Der hier vorgeschlagene Weg bietet aber mehr Freiraum bei der sukzessiven Erweiterung des Start-Makros.

Modul umbenennen

Als nächstes wird *Modul1* umbenannt. Markieren Sie im Projekt-Fenster mit einem Klick das neu eingefügte Modul, doppelklicken Sie dann im Eigenschaften-Fenster bei *Name* auf *Modul1* und überschreiben Sie den Namen durch *Formularinhalte*. Im Projektfenster hat sich der Name ebenfalls geändert, siehe Bild unten.

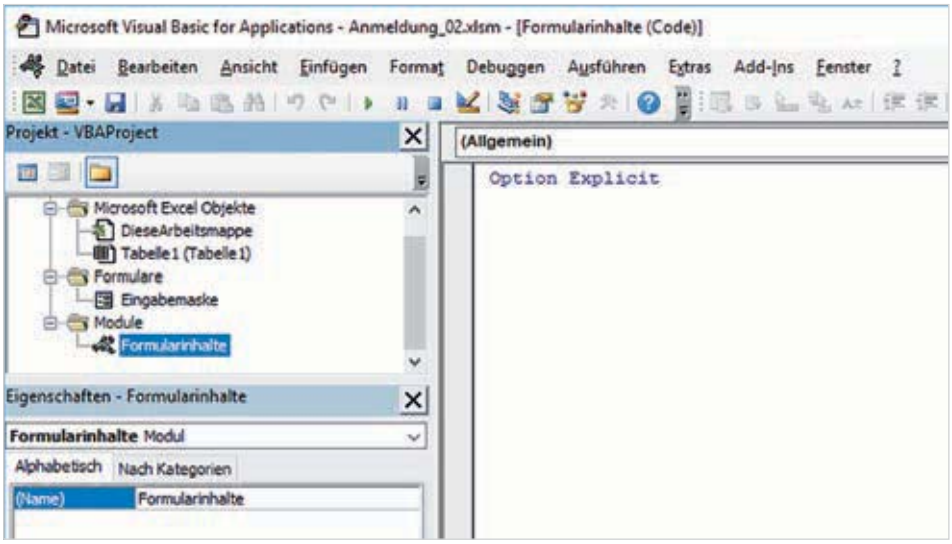


Bild 8.15 Das eingefügte Modul wurde umbenannt

Wir wechseln per Mausklick ins Code-Fenster. Im nächsten Schritt muss die bereits erstellte Eingabemaske aufgerufen werden, und zwar möglichst auch aus dem Tabellenblatt (Grundansicht) heraus.

Erklärung: Makro ist eine Kurzbezeichnung für Prozeduren, eine Aneinanderreihung mehrerer Anweisungen (Einzelhandlungen). Jedes Makro bzw. jede Prozedur wird als Programmschritt – als „Subroutine" – behandelt und daher mit der Anweisung *Sub()* eingeleitet und mit *End Sub* abgeschlossen. Dieser äußerliche Aufbau wird Prozedurrumpf genannt. Sobald wir die erste Zeile unseres neuen Makros geschrieben

haben und mit der Enter-Taste beenden, wird die Prozedur automatisch um *End Sub* zum Prozedurrumpf ergänzt.

Bild 8.16 Der Prozedurrumpf wird automatisch erzeugt

```
Option Explicit

Sub Eingabemaske_starten()

End Sub
```

Um unsere Eingabemaske aus der VBA-Umgebung bzw. aus einem Tabellenblatt heraus aufzurufen, müssen wir den entsprechenden Ausführungsbefehl in die Prozedur schreiben.

Das Objekt, das wir ansprechen wollen, hat den Namen *Eingabemaske* erhalten und es soll angezeigt werden, also *Show*. In der Befehlszeile werden Objekt und *Methode* mit einem Punkt verbunden. Oder anders ausgedrückt: die *Methode Show* wird auf das *Objekt Eingabemaske* angewandt:

```
Eingabemaske.Show
```

Nach Eingabe des Punkts öffnet sich eine Liste, die Ihnen Hilfe anbietet, indem sie zum angesprochenen Objekt passende Eigenschaften und Methoden auflistet. Hier findet man auch *Show*. Schneller geht's, wenn Sie auch noch die ersten Zeichen eintippen.

Bild 8.17 Liste Autovervollständigen

Bild 8.18 Das Makro zum Anzeigen des Formulars

```
Sub Eingabemaske_starten()

    Eingabemaske.Show

End Sub
```

Tipp: Sobald Sie die ersten Buchstaben einer Eigenschaft oder Methode eingegeben haben, steht über die Tastenkombination Strg + Leertaste die automatische Vervollständigung (Microsoft: IntelliSense) zur Verfügung. Wählen Sie mit der Pfeiltaste nach oben/unten einen Befehl aus und betätigen Sie zum Einfügen die Tab-Taste.

Weiter zum ersten Erfolgserlebnis: Prozedur starten

- Bleiben Sie mit dem Cursor innerhalb des Makros, irgendwo zwischen *Sub* und *End Sub*.

- Starten Sie Ihr erstes Makro über den grünen Pfeil in der Menüleiste oder durch Drücken der Taste *F5*.
- Die Eingabemaske erscheint!

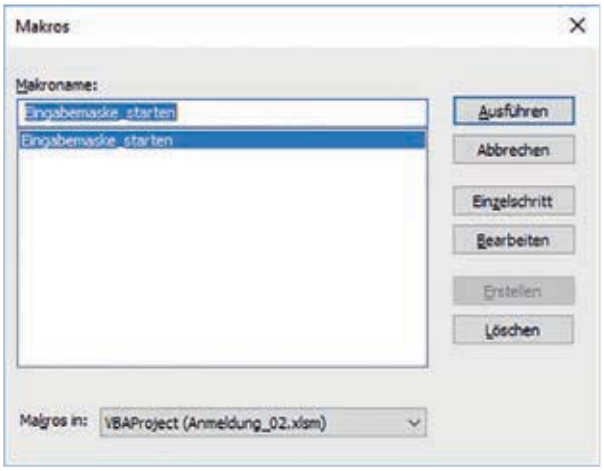
Hinweis: Befindet sich der Cursor außerhalb des Makros oder zwischen zwei Makros, öffnet sich das unten abgebildete Fenster *Makros* und bietet die Auswahl aus vorhandenen Makros an. Nach erfolgter Auswahl und der Schaltfläche *Ausführen* kommt man zum gleichen Ergebnis/Erlebnis wie oben beschrieben, nur etwas umständlicher.

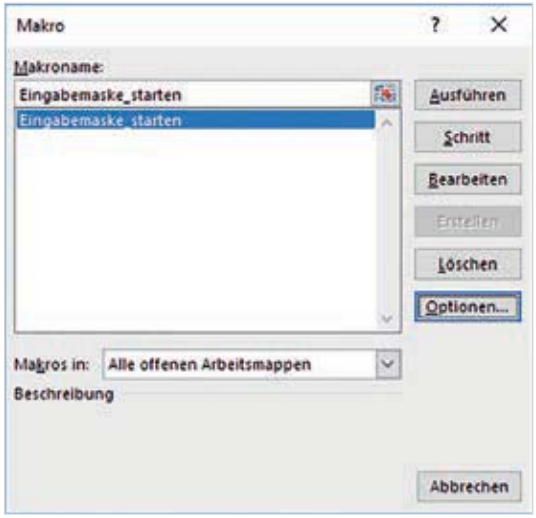
Siehe auch Kapitel 2 zum Thema Makros aufzeichnen und ausführen.

- Dieses Fenster kann auch über *Extras* ▸ *Makros* geöffnet werden. Über die Schaltfläche *Bearbeiten* kommt man wieder zum Code-Fenster.
- Aus dem Arbeitsblatt heraus gelangen Sie über *Entwicklertools* ▸ *Makros* an die vorhandenen Makros.

Bild 8.19 Makro auswählen und starten

Bild 8.20 Makros mit Optionen (Entwicklertools)





Eingabemaske starten

Tastenkombination zuweisen

Beim Anzeigen aus dem Arbeitsblatt heraus über *Entwicklertools* ▸ *Makros* (Bild 8.20) können Sie über die Schaltfläche *Optionen…* eine zusätzliche Beschreibung eingeben und dem Makro eine Tastenkombination zuweisen. Nehmen Sie die Eintragungen wie im Bild 8.21 vor und bestätigen Sie mit *OK*. Im Makro-Fenster taucht nun die Beschreibung zum Makro auf – eine hilfreiche Option.

Hinweis: In die Tastaturkombination können Sie jeden beliebigen Buchstaben in Groß- oder Kleinschreibung einbeziehen. Standardmäßige zugewiesene Funktionen werden während der Anwendung außer Kraft gesetzt, solange die Arbeitsmappe mit dem Makro geöffnet ist.

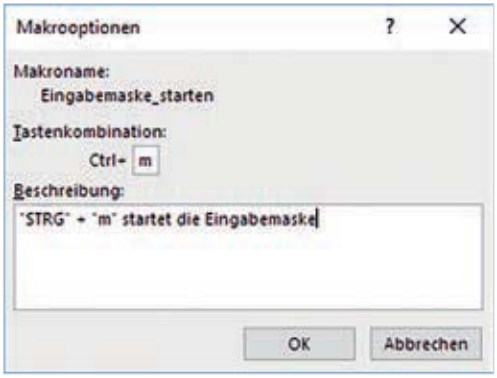
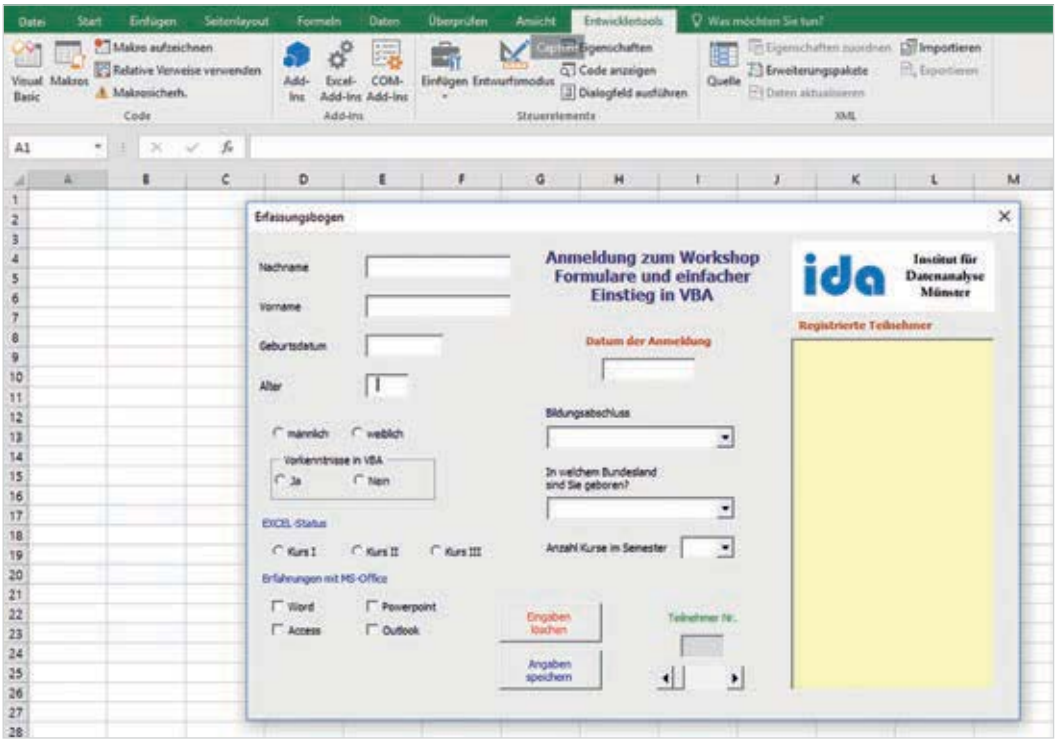


Bild 8.21 Festlegen einer Tastenkombination zum Aufruf des Makros

Tipp: Die Kombinationen Strg + j und Strg + m sind aktuell (Excel 2013 und 2016) nicht zugewiesen und eignen sich daher zum Aufruf von Makrobefehlen ohne andere Funktionen zu verdrängen.

Die Schaltfläche *Abbrechen* schließt das Fenster *Makros* und bringt uns wieder in die Tabellenansicht zurück. Dort testen wir sofort unsere festgelegte Tastenkombination Strg + m und erhalten die Eingabemaske vor der Tabelle.

Bild 8.22 Test der Tastenkombination



Eingabemaske automatisch beim Öffnen der Mappe anzeigen

Mit dieser soeben festgelegten Tastenkombination ist das Aufrufen der Maske aus der Tabelle jederzeit möglich. Weitere Programmiertechniken erlauben es, sofort beim Öffnen der Arbeitsmappe die Maske automatisch zu starten - ebenfalls durch Aufruf des entsprechenden Makros. Gehen Sie dazu wie folgt vor:

- Aktivieren Sie im Projekt-Fenster mit Doppelklick *Diese Arbeitsmappe*.
- Wählen Sie im Code-Fenster mit Klick auf den Dropdown-Pfeil *Workbook* aus. Rechts daneben erscheint automatisch das Ereignis *Open* und der dazugehörige Prozedurrumpf wird im Code-Fenster erzeugt.

Bild 8.23 Workbook auswählen

Charakteristische Zeichenfolgen (Bestellcode, Kennzeichen, ID-Code)

In manchen Fällen sind für Bestell- oder Rechnungsnummern strukturierte Zeichenfolgen zu überprüfen. Im Bild ist im Feld *Eingabe1* die Eingabe eines Buchstabens zwischen A und Z sowie nach dem Bindestrich eine dreistellige Zahl erforderlich, z. B. A-123. Das Feld *Eingabe2* erfordert zwei Buchstaben sowie eine dreistellige Zahl.

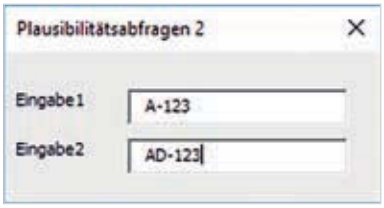


Bild 8.64 Zwei unterschiedliche charakteristische Zeichenfolgen werden erwartet (UserForm2)

Dazu können im Vergleichsstring Wertebereiche durch vorgegebene Zeichen in [ ] eingegrenzt werden. Einzelne Zeichen werden einfach eingegeben, Bereiche können mit Bindestrich angegeben werden. Die Groß- und Kleinschreibung ist zu beachten, es sei denn beide Zeichen werden explizit durch [A-X,a-x] erlaubt.

```
Private Sub Eingabe1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
' strukturierte Zeichenfolge prüfen

    If Me.Eingabe1.Value Like "[A-X]-###" Then
        MsgBox "Eingabestruktur korrekt"
    Else
        MsgBox "Fehlerhafte Eingabe!"
        Cancel = True
        Me.Eingabe1.Value = ""
    End If
End Sub
```

```
Private Sub Eingabe2_Exit(ByVal Cancel As MSForms.ReturnBoolean)
' strukturierte Zeichenfolge prüfen
    If Me.Eingabe2.Value Like "[A-X][A-X]-###" Then
        MsgBox "Eingabestruktur korrekt"
    Else
        MsgBox "Fehlerhafte Eingabe!"
        Cancel = True
        Me.Eingabe2.Value = ""
    End If
End Sub
```

Bild 8.65 Prozeduren für Zeichenkettenabfrage in UserForm2

IBAN-Struktur für Deutschland prüfen

In Deutschland hat jede IBAN (International Bank Account Number) genau 22 Stellen. Im Papierformat muss diese Kennung in 4er-Blöcken durch Leerzeichen getrennt geschrieben werden. Sonderzeichen oder Kleinbuchstaben dürfen nicht enthalten sein. Im elektronischen Format entfallen die Leerzeichen. Die ersten beiden Zeichen repräsentieren den Ländercode (z. B. DE für Deutschland) gefolgt von einer zweistelligen Prüfsumme für die gesamte IBAN, 8 Stellen für die Bankleitzahl und eine 10-stellige Kontonummer (fehlende Stellen werden von vorn mit Nullen aufgefüllt).

Bild 8.66 IBAN prüfen auf zwei unterschiedliche Formate

Bild 8.67 Prozeduren zur IBAN-Überprüfung in UserForm2

```
Private Sub Eingabe1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
' IBAN prüfen (Elektronisches Format: keine Leerzeichen, 22 Stellen )

    If Len(Me.Eingabe1.Text) <> 22 Then
        MsgBox "IBAN zu kurz/lang - ohne Leerzeichen!"
    Else
        If Me.Eingabe1.Value Like "[A-X][A-X]#*" Then
            MsgBox "Eingabestruktur korrekt"
        Else
            MsgBox "Fehlerhafte Eingabe!"
            Cancel = True
            Me.Eingabe1.Value = ""
        End If
    End If
End Sub
```

```
Private Sub Eingabe2_Exit(ByVal Cancel As MSForms.ReturnBoolean)
' IBAN prüfen (Papierformat: 4er Blöcke und Leerzeichen, 27 Stellen )

    If Len(Me.Eingabe2.Text) <> 27 Then
        MsgBox "IBAN zu kurz/lang - Leerzeichen vergessen?"
    Else
        If Me.Eingabe2.Value Like "[A-X][A-X]## #### #### #### #### ##" Then
            MsgBox "Eingabestruktur korrekt"
        Else
            MsgBox "Fehlerhafte Eingabe!"
            Cancel = True
            Me.Eingabe1.Value = ""
        End If
    End If
End Sub
```

Eingaben einschränken

Neben der Abfrage von Eingabemustern gehört zur Plausibilitätsprüfung auch die Beschränkung der Eingabewerte auf zulässige Zeichen zur Vermeidung oder Reduzierung von Fehleingaben und deren Folgen.

Dazu lässt sich das Ereignis *KeyPress* des Textfeldes nutzen. Es tritt auf, wenn der Benutzer eine Taste oder Tastenkombination betätigt und wieder loslässt, während das Textfeld den Fokus hat und liefert den ASCII-Wert der betätigten Taste (*KeyAscii*).

Beliebige Buchstaben erlauben

Über eine Abfrage der ASCII-Werte des Tastaturcodes lassen sich beliebige Buchstaben freigeben.

Bild 8.68 Vorgabewerte erlaubter Tasten für UserForm3

```
Private Sub Eingabe1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
' Nur die Zeichen A-X,a-x,-,Ä,Ö,Ü,ä,ö,ü akzeptieren

    Select Case KeyAscii
        Case 65 To 90, 96 To 122, 45, 196, 214, 220, 228, 246, 252
            'MsgBox "OK"
        Case Else
            MsgBox "verboten"
            KeyAscii = 0
    End Select

End Sub
```

Nur Ziffern zulassen (Ganze Zahlen, Kommazahlen)

Die Freigabe für einzelne Ziffern betrifft die Zahlen 0 bis 9 (ASCII 48 ... 57) und ggf. auch das Dezimaltrennzeichen Komma (ASCII 44). Alternativ kann die Funktion *IsNumeric* verwendet werden.

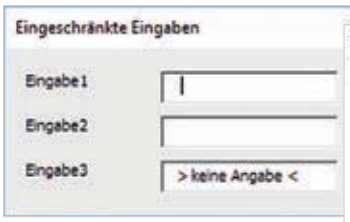
```
Private Sub Eingabe2_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
' Nur die Zeichen 0-9 sowie das Trennzeichen Komma akzeptieren

    Select Case KeyAscii
        Case 48 To 57, 44
            'MsgBox "OK"
        Case Else
            MsgBox "verboten"
            KeyAscii = 0
    End Select

End Sub
```

Bild 8.69 Es werden nur Ziffern (Zahlen) zugelassen

Leertaste als Ersatz für Text oder Sonderzeichen

Über das *KeyPress*-Ereignis lassen sich auch bestimmte Tasten der Tastatur abfangen und für Sonderzeichen oder spezielle Zeichenketten verwenden. Beispielsweise um, wie im Bild unten, beim Betätigen der Leertaste (ASCII 32) *>Keine Angabe<* anzuzeigen.

Bild 8.70 Leertaste veranlasst das Einfügen einer Zeichenkette




```
Private Sub Eingabe3_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
' Leer-Taste veranlasst den Eintrag "> keine Angabe <"

        If KeyAscii = 32 Then
            Eingabe3.Value = "> keine Angabe <"
            'weiter
            Eingabe1.SetFocus
        End If

End Sub
```

Auf diese Weise lassen sich für bestimmte Textfelder ganz individuelle Lösungen herausarbeiten. Das unten abgebildete Beispiel fügt bei Betätigen der Leertaste das Sonderzeichen | ein.

Bild 8.71 Leertaste fügt Sonderzeichen in einen String ein


```
Private Sub Eingabe3_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
' Leer-Taste fügt (wiederholt) das Pipe-Symbol(124) hinzu

        If KeyAscii = 32 Then
            Eingabe3.Value = Eingabe3.Value & Chr(124)
        End If

End Sub
```

Hinweis: Das *KeyPress*-Ereignis reagiert auf alle druckbaren Tastaturzeichen sowie die Enter- und die Rückschritt-Taste. Für Tasten, die vom *KeyPress*-Ereignis nicht erkannt werden, z. B. Esc-Taste oder Entf-Taste muss das Ereignis *KeyDown* verwendet werden.

Beispieldatei: Plausibilitätsprüfung.xlsm

8.4 Daten aus der Eingabemaske in die Tabelle übertragen

Wenn alle Bedingungen erfüllt sind und die Textfelder *Nachname*, *Gender* und VBA-Vorkenntnisse ausgefüllt wurden, können die Einträge aus der Eingabemaske in das Arbeitsblatt übertragen werden. Auf diesem Weg lassen sich über Abfragen (If… Then) auch entsprechende Abkürzungen oder Symbole in die entsprechenden Spalten eintragen.

Für die Prozedur mit den Anweisungen zum Speichern der Maskeninhalte wird sinnvollerweise ein neues Modul mit dem Namen *Datentransfer* eingefügt, die Prozedur selbst erhält den Namen *Daten_speichern*. Das Speichern erfolgt im Arbeitsblatt *Tabelle1*, dort werden ab Zeile 2 alle Angaben spaltenweise abgelegt. Die erste Zeile bleibt vorerst frei für Spaltenüberschriften.

Den ersten Datensatz aus der Eingabemaske übernehmen

Bild 8.72 Makro zum Daten speichern (in Zeile2)

```
Sub Daten_speichern()
'Daten aus der Eingabemaske als Zahlen mit CDbl() in Tabelle1 ablegen
'Aufruf über Schaltfläche "Angaben speichern"

    Worksheets("Tabelle1").Activate

    With Eingabemaske
        'Angaben zur Person Spate A - D
        Range("A2").Value = .Nachname.Value
        Range("B2").Value = .Vorname.Value
        If IsDate(.GebDat.Value) Then
            Range("C2").Value = CDate(.GebDat.Value)
        Else
            Range("C2").Value = "?"
        End If
        Range("D2").Value = CDbl(.Alter.Value)
        'Gender in Spalte E
        If .Gender_m Then Range("E2").Value = "m"
        If .Gender_w Then Range("E2").Value = "w"
        'VBA-Vorkenntnisse in Spalte F
        If .Vorkenntnisse_ja Then
            Range("F2").Value = "ja"
        Else
            Range("F2").Value = "nein"
        End If
        'EXCEL-Status in Spalte G
        If .Kurs1 Then Range("G2").Value = "I"
        If .Kurs2 Then Range("G2").Value = "II"
        If .Kurs3 Then Range("G2").Value = "III"
        'Erfahrungen mit MS-Office in Spalte H - K
        If .MS_Word Then Range("H2").Value = "x"
        If .MS_Access Then Range("I2").Value = "x"
        If .MS_PPT Then Range("J2").Value = "x"
        If .MS_Outlook Then Range("K2").Value = "x"
        'Bildungsabschluss
        Range("L2").Value = .Bildung.Value
        'Bundesland
        Range("M2").Value = .Bundesland.Value
        'Kurse im Semester
        Range("N2").Value = .Kurse.Value
    End With

End Sub
```

Der zeitliche Ablauf im Überblick

1. Auswählen des Arbeitsblatts *Tabelle1* mit *Activate* oder *Select*,
2. Nachname und Vorname werden aus der Eingabemaske direkt in die Zellen A2 und B2 übernommen,
3. Das Geburtsdatum wird nach einer Plausibilitätsprüfung übernommen oder durch ? ersetzt,
4. Gender wird mit zweimaliger If…Then-Abfrage übernommen als m oder w.
5. VBA-Vorkenntnisse werden mit einer If…Then…Else-Abfrage als ja oder nein übernommen.
6. Kurs1 bis Kurs3 werden entsprechend mit den römischen Zahlen I, II, III in die Spalte G übernommen,
7. MS-Office-Erfahrungen jeweils in einer eigenen Spalte mit x,
8. Bildung, Bundesland und Anzahl Kurse werden wieder direkt übernommen.

Angaben speichern

Die Schaltfläche *Angaben speichern* wurde bereits mit etlichen Sicherheitsabfragen angelegt. Zum Speichern muss in der letzten Zeile noch der Aufruf der Prozedur *Daten_speichern* eingefügt werden.


```
Private Sub cmd_speichern_Click()
'Datenübergabe an Tabelle nach Pflichfeldeingaben

    'Abfrage nach Nachname
    If Me.Nachname.Value = "" Then
        MsgBox "Bitte den Nachnamen eingeben, sonst Speichern nicht möglich"
        Exit Sub
    End If
    'Abfrage der Optionsfelder
    If Me.Gender_neutral Then
        MsgBox "Bitte legen Sie sich fest: m/w"
        Exit Sub
    End If
    If Me.Vorkenntnisse_neutral Then
        MsgBox "Bitte geben Sie Ihre VBA-Vorkenntnisse an"
        Exit Sub
    End If

    'Speichern der Feldinhalte der Eingabemaske
    Daten_speichern

End Sub
```

Bild 8.73 Makro zum Daten speichern einbinden (letzte Zeile)

Im Bild 8.74 auf der nächsten Seite ein erster Test und Überprüfung der korrekten Eingabefolge.

```
Public Const Leerfeld As String = " ... "
Public nummer As Integer

Sub starte_maske()
    nummer = 1
    Maske.Show
End Sub

Sub anzeigetext(nr As Integer, block As String)
    Worksheets("Tabelle1").Activate
    Maske.Textfeld.Value = Range("A" & nr).Value & block & Range("C" & nr).Value
End Sub
```

Bild 9.19 Fragenbildung

Zuvor wird eine Anweisung/Aufforderung im Beschriftungsfeld *Label1* angezeigt.

```
Private Sub UserForm_Initialize()
    Me.Label1.Caption = "Bitte wählen Sie die korrekte Ergänzung"
    anzeigetext nummer, Leerfeld
End Sub
```

Bild 9.20 Beim Initialisieren der UserForm

Um eine Voreinstellung der Optionsfelder für *mir* und *mich* zu vermeiden, wurde ein drittes Optionsfeld (neutral) eingefügt, mit den Voreinstellungen *Value = True* und verdeckt (*Visible = False*). Das zweite Beschriftungsfeld (*Label2*) ist sichtbar, aber ohne Inhalt (*Caption*). Es wird als Hinweisfeld für Dativ/Akkusativ zur Anwendung kommen.

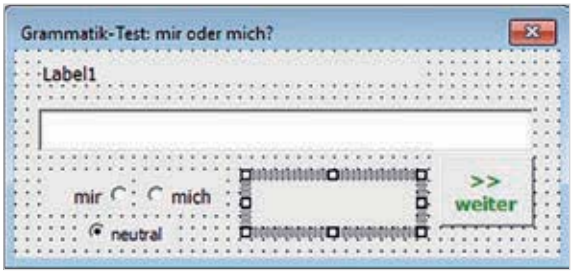
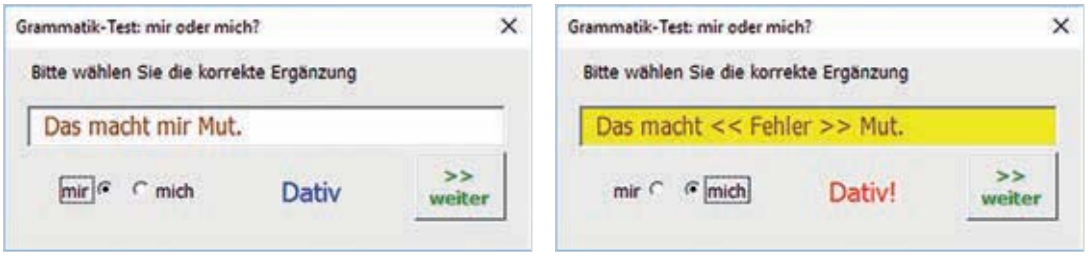


Bild 9.21 Formularansicht mit verdecktem Optionsfeld und markiertem Beschriftungsfeld Label2

Die Kontrolle, ob die Entscheidung korrekt war, erfolgt mit dem Klick in ein Optionsfeld. Die richtige Lösung wird angezeigt oder bzw. ein Fehler mit Hinweis auf den zu wählenden Fall bei einer falschen Antwort.

Bild 9.22 Die Anzeige nach richtiger und falscher Eingabe



Der Klick auf *weiter* bringt die nächste Frage zur Anzeige. Die vorliegende Tabelle kann nach eigenem Belieben erweitert oder verändert werden und kann auch, wie zu Beginn wieder verdeckt werden. Die Formulare und Makros finden Sie in der Datei *Beispiel_Grammatiktest.xlsm*.

Datei: Beispiel_Grammatiktest.xlsm

9.5 Formular mit Multiple-Choice-Fragen

Die meisten Tests/Prüfungen basieren auf Multiple-Choice-Verfahren. Dabei sind alle Antwortkombinationen zwischen „Keine Antwort ist richtig“ und „Alle Antworten sind richtig“ möglich. Das folgende Beispiel überprüft, ob die gewählten Antworten der korrekten Kombination entsprechen.

Bild 9.23 Formular für Multiple-Choice-Fragen

Kontrolle der Antworten

Zur Überprüfung der richtigen Antwortkombination gibt es mehrere Wege. Wir haben uns hier für die Zuordnung von Zahlenwerten entschieden, aus deren Kombination eindeutig die gesetzten Antworten ableitbar sind. Werden den vier Antwortmöglichkeiten A bis D beispielsweise die Zahlen 2, 4, 8 und 16 zugeordnet, lässt sich aus der Wertekombination eine Prüfsumme ermitteln, die der Lösungssumme entspricht. Die fünfte Antwort E „Keine Antwort ist richtig“ deaktiviert alle anderen Antworten und die setzt die Lösungssumme auf 1. Ist die Lösungssumme 0, dann bedeutet das, dass keine Antwort gegeben wurde und ein entsprechender Hinweis wird angezeigt.

Bild 9.24 Antwortkombinationen

Antwortkombination					Lösung	
D	C	B	A	E	Wert	
$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	dezimal	
0	0	0	1	0	2	
0	0	1	0	0	4	
0	0	1	1	0	6	
0	1	0	0	0	8	
0	1	0	1	0	10	
0	1	1	0	0	14	
...	...	...	...		...	
1	1	1	1	0	30	
0	0	0	0	1	1	keine Antwort ist richtig
0	0	0	0	0	0	keine Antwort gegeben

Um auch optisch die richtigen Antworten hervorzuheben – wie ja oder nein – stehen symbolisch die Zahlen 1 oder 0 in den Tabellenspalten, die für sich jeweils einen Potenzwert der Zahl 2 repräsentieren. (Eine kleine „Spielerei" im Binärbereich – die direkte Wertezuordnung ist selbstverständlich auch möglich.)

Bild 9.25 Fragen, Antwortmöglichkeiten und Lösung sind in Tabelle1 hinterlegt

Nr	Frage	Antwort A	Antwort B	Antwort C	Antwort D	A	B	C	D	E (keine)	Summe
1	In welchen Produkten kann	MS-Word	MS-Excel	MS-Powerpoint	MS-Access	1	1	1	1		30
2	Welche Elemente gehören	Kombinationsfeld	Bild (Anzeigen)	Umschaltflächen	Bezeichnungsfelder		1	1			12
3	Was bewirkt Application.Sc	Der Bildschirm wi	Das Programm kann be	Der Bildschirm wird für	Das Flackern beim Anzeigenwechsel wird		1		1		20
4	Was bedeutet die Vorgabe	Variablen müsser	Der Index von Arrays b	Es kommt grundsätzlicl	Variablen werden öffentlich	1					2
5	Wie wird eine lokale Variat	Mit dem Zusatz "li	Mit dem Zusatz "privat	innerhalb des Makros	Außerhalb des Makros			1			8
6	Wodurch sind Exceldateien	Beim Öffnen erfo	Die Dateierweiterung	Die dateierweiterung	Es besteht äußerlich kein Un	1	1				6
											0

In den Spalten H bis L bestimmt eine 1 oder eine 0 (leer) die Wertigkeit der 5 Antwortmöglichkeiten. Dazu werden den Kontrollkästchen *Antwort_A* bis *Antwort_E* Potenzwerte der Zahl 2 zugeordnet und über eine Summenformel in Spalte M übernommen.

Bild 9.26 Summenbildung in Spalte M, Zeile 2

WENN =(H2*2)+(I2*4)+(J2*8)+(K2*16)+L2

Nr	A	B	C	D	E (keine)	Summe	
1	1	1	1	1		=(H2*2)+(I2*4)+(J2*8)+(K2*16)+L2	
2		1	1			12	UK 11
3		1		1		20	UK 11

Diese Vorgehensweise erlaubt eine eindeutige Interpretation aller Antwortkombinationen. Zur Verdeutlichung der Summenbildung bei unterschiedlichen Kombinationen befindet sich in der Eingabemaske ein quadratisches rötlich hinterlegtes Textfeld.

Die Dialogfelder mit den Emotions bei korrekter bzw. falscher Eingabe sind dem Beispiel aus Punkt 9.3, Formular als VBA-Wissenstest, entnommen.

Formuar testen

Die Formulare und Makros finden Sie in der Datei *Beispiel_MultiplChoice.xlsm*. Das Multiple-Choice-Formular startet automatisch beim Öffnen der Datei vor einem hellgrauen Hintergrund. Zum Beenden des Tests muss zuerst das Formular beendet werden. Die Schaltfläche *Beenden* in der linken oberen Ecke schließt die Datei.

Datei: Beispiel_MultipleChoice.xlsm

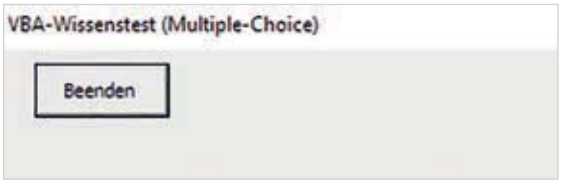


Bild 9.27 Datei schließen

Doch wie kommt man „Backstage" in die Entwicklungsumgebung? Wir möchten Ihnen das nicht vorenthalten, denn schließlich sollen die hier gezeigten Lösungsansätze Sie motivieren. Ganz bestimmt finden Sie auch noch Verbesserungsmöglichkeiten. Zur Anwendung kommen ein paar einfache „dirty tricks".

- Beim Öffnen der Datei werden zwei UserForms angezeigt: die Hintergrundmaske mit der Schaltfläche *Beenden* als *vbModeless* und darüber *Maske2* mit den Multiple-Choice-Fragen.

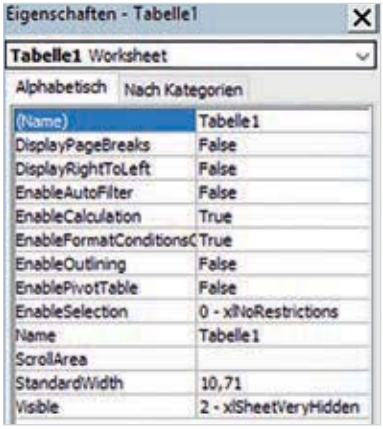
Bild 9.28 Die beiden Masken starten

```
Sub maske_starten()

    Hintergrundmaske.Show vbModeless
    antwort = ""
    Anzahl_Fragen = 0
    Anzahl_richtig = 0
    Anzahl_falsch = 0
    frage_generieren 2
    Maske2.Show

End Sub
```

Achtung: Die Hintergrundmaske ist übergroß angelegt: Height 1800, Width 2400. Daher befindet sich die *Schließen*-Schaltfläche *x* zum Beenden der Maske rechts außerhalb des sichtbaren Bildschirmbereichs, was der Anwender (sonst) nicht ahnt. Verschieben Sie also die graue Hintergrundmaske so lange nach links bis Sie über das *x* die Maske schließen können.

Bild 9.29 Eigenschaften von Tabelle1

- Wenn Sie sich auf diese Weise Zugang zu den Tabellen verschafft haben, finden Sie nur das Tabellenblatt *Oberfläche* mit einheitlich grauen Zellen vor. Auch diese könnte schon als Hintergrund ausreichen. Die eigentliche Tabelle mit den Fragen und Lösungen wurde aktiv versteckt, siehe Bild, Eigenschaften.



Projekt im VBA-Editor anzeigen

Wir treffen uns Backstage (Tastenkombination Alt + F11). Der VBA-Editor wird auch angezeigt, doch die Anzeige des aktuellen Projekts ist durch ein Passwort geschützt. – Es lautet „Passau“ (wie sonst?). Das Projekt wurde im VBA-Editor über den Menübefehl *Extras* ▸ *Eigenschaften von VBAProjekt* geschützt (siehe auch Seite 124).

Bild 9.30 Kennwort zum Anzeigen des Projekts eingeben