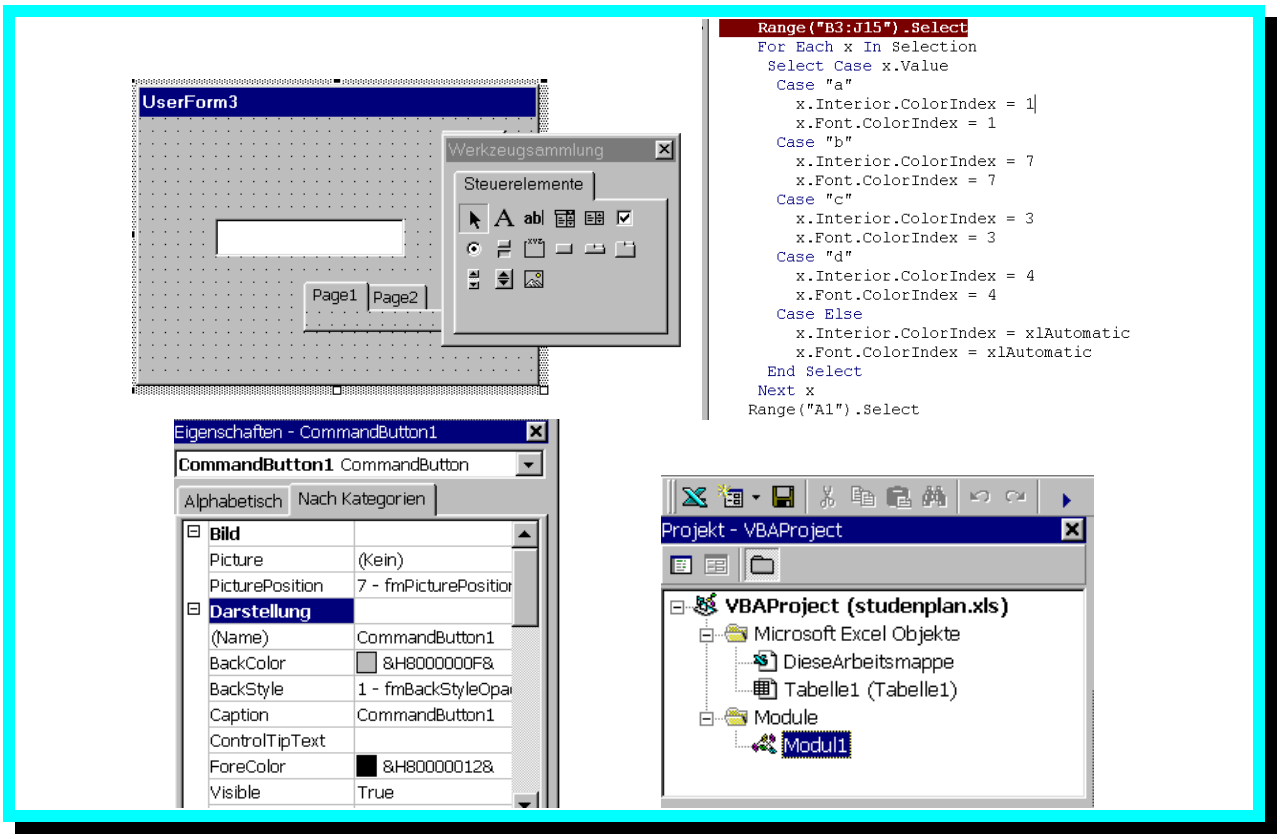
FernUniversität
Gesamthochschule in Hagen
Universitätsrechenzentrum



Johannes Gogolok
Abt. Wiss. Anwendungen

URZ B/012/9911

VBA – Programmierung mit Excel

Grundlagen

REIHE: INTEGRIERTE SYSTEME

VBA – Programmierung mit Excel

Grundlagen

November 1999

Inhaltsverzeichnis

Verzeichnis der Abbildungen

1 Einleitung

EXCEL ist bereits in seiner Standardform ein mächtiges Instrument zur Tabellenkalkulation mit vielen Formen und Funktionen der Kalkulation, der Datenanalyse und –präsentation. In Verbindung mit VBA (VISUAL BASIC FOR APPLICATIONS) wird es zu einem noch mächtigeren Entwicklungssystem, welches es dem Anwender erlaubt, das Spektrum des Programms an die eigenen Bedürfnisse anzupassen und die Anwendungen beträchtlich zu erweitern.

VBA ist allerdings ein sehr umfangreiches Instrument, dessen Beherrschung ständige Übung erfordert. Es enthält sehr viele Sprachelemente. Deren vollständige Behandlung würde den Rahmen dieser Broschüre sprengen.

Deshalb erhebt diese Unterlage nicht den Anspruch, VBA vollständig zu behandeln, sondern nur jene Arbeitsanweisungen vorzustellen, die zur Steuerung von VBA-Programmen und die Erstellung benutzerdefinierter Abläufe benötigt werden. Sie soll den Anwender in die Lage versetzen, das Grundgerüst eines VBA-Programms erstellen zu können und dieses um Anweisungen zu ergänzen, die eine der zu lösenden Aufgabe angepaßte Verarbeitung mittels eigener Anwendungen gestatten.

Zwar sind die interaktiven Funktionen von EXCEL immer flexibler und umfangreicher geworden, parallel dazu hat sich aber auch die Programmierbarkeit entwickelt. Aus einer recht einfachen Makro – Sprache ist eine recht umfangreiche, objektorientierte Programmiersprache geworden, die den Vergleich mit anderen Entwicklungssystemen nicht fürchten muß.

Das VBA ist in EXCEL in englischer Sprache implementiert. Die EXCEL – Funktionen sind jedoch in der deutschen Version von EXCEL in deutscher Sprache definiert, obwohl VBA sie im Programmcode auf Englisch erwartet.

Kleiner Tip schon an dieser Stelle:

Im leeren Tabellenblatt den Makrorekorder starten, Funktion in eine Zelle eintragen und den aufgezeichneten Programmcode ins Programm übertragen.

Für die erfolgreiche Arbeit mit dieser Unterlage sind mindestens gute Grundkenntnisse von WINDOWS und EXCEL erforderlich. Vorausgesetzt werden auch zumindest Grundkenntnisse des VB (*Visual Basic*), weil VB den Sprachkern von VBA liefert.

Im Text der Unterlage befinden sich am rechten Textrand Hinweiszeichen mit der folgenden Bedeutung:

 Tip aus der Praxis

 Hinweis

 Wichtige Hinweise zur Makros, Prozeduren und Funktionen

 Makrobeispiele

Ein Hinweis in eigener Sache:

Mein Dank gilt an dieser Stelle meinem Kollegen Martin Kohl für die Hilfe bei der Textkorrektur und insbesondere meinem Sohn Sebastian für die Geduld bei der Kontrolle und Testen der Programmbeispiele und der Lösungen der Übungsaufgaben.

2 Das erste Excel-Programm – ein aufgezeichnetes Makro

EXCEL verfügt, wie alle anderen Office-Programme, über einen Makro - Recorder, mit dem sich Aktionen innerhalb der Tabellenblätter oder Arbeitsmappen aufzeichnen lassen. Die dabei entstehenden Makros sind eine Vorstufe der VBA - Programme. Sie werden vom Recorder in den VBA – Code umgesetzt und können nachträglich beliebig erweitert werden. In vielen Fällen ist ein aufgezeichnetes Makro ein Grundgerüst für ein VBA – Programm oder liefert Teillösungen für ein solches.

Aufgezeichnete Makros stoßen jedoch recht schnell an ihre Grenzen, z.B.:

- Es lassen sich keine Schleifen verwenden, die eine mehrfache Ausführung einer Aktion erlauben.
- An Bedingungen gebundene Sicherheitsabfragen lassen sich beim Aufzeichnen nicht per Mausklick definieren, was dazu führt, daß vor dem Start des Makros alle notwendigen Vorkehrungen getroffen werden müssen, welche den korrekten Ablauf garantieren, z.B. die richtige Zelle markiert oder den Cursor richtig positioniert.
- Der Automatisierungsgrad eines Makros ist sehr gering, da i.d.R. nur seriell ablaufende Prozesse aufgezeichnet werden.

Ein aufgezeichnetes Makro ist damit ein recht nützliches Werkzeug, welches jedoch nur eine Teilautomatisierung von Abläufen gestattet, die „Vollautomatik“ muß programmiert werden.

2.1 Makro aufzeichnen

2.1.1 Eine Beispieltabelle

Als Beispiel soll ein Makro aufgezeichnet werden, welches das Einfügen einer Zeile in eine bestehende Tabelle incl. dazugehöriger erforderlicher Formeln ermöglichen soll.

Die Tabelle soll der Verbuchung von Einnahmen und Ausgaben dienen. Für jede neue Eingabe soll eine neue Zeile erzeugt werden.

	A	B	C	D	E	F	G
1	BelegNr	Datum	MwSt Satz	Einnahme brutto	Einnahme netto	Ausgabe brutto	Ausgabe netto
2							
3					=D3/(1+C3)		=F3/(1+C3)
4							

Abbildung 1: Tabelle für die Makroaufzeichnung

In der Abbildung wurden die benötigten Formeln sichtbar gemacht.

Das aufzuzeichnende Makro soll nun die folgenden Aufgaben erledigen:

- Eine neue Zeile einfügen (hier Zeile 3).
- In die Spalte E der neuen Zeile die Formel =D3/(1+C3) einfügen.
- In die Spalte G der neuen Zeile die Formel =F3/(1+C3) einfügen.

➡ Vor der Aufzeichnung eines Makros muß ein definierter Anfangszustand des Tabellenblattes hergestellt werden, beispielsweise eine bestimmte Zelle muß markiert werden. Die Kontextanforderungen sollten möglichst minimal sein.

Für das Beispiel gilt, daß der Cursor in einer Zeile stehen muß, über der eine neue Zeile eingefügt werden soll.

✘ Die aufzuzeichnenden Aktionen sollten vor der Aufzeichnung getestet werden, um unnötige Korrekturen im aufgezeichneten Makro zu vermeiden bzw. um nicht unnötig mehrfach aufzeichnen zu müssen.

2.1.2 Die Makroaufzeichnung

Die Makroaufzeichnung wird über die Funktionskombination EXTRAS / MAKRO / AUFZEICHNEN aufgerufen. Alternativ kann der Schalter  der Symbolleiste VISUAL BASIC benutzt werden (einzublenden über ANSICHT / SYMBOLLEISTEN / VISUAL BASIC).

Im nach dem Aufruf der Aufzeichnung erscheinenden Dialogfeld NEUES MAKRO AUFZEICHNEN sind vor dem Start noch einige Angaben zum Makro zu tätigen:

Abbildung 2: Makrodefinition

- Der *Makroname* (hier *N_Zeile*), maximal 255 Zeichen lang. Der Name muß mit einem Buchstaben beginnen, Leerzeichen und Bindestriche sind nicht erlaubt. Umlaute im Namen sind zulässig.
- Eine *Tastenkombination* zum Starten des Makros (andere Startalternativen – siehe weiter im Text). Bei der Eingabe des Buchstabens N im obigen Beispiel wurde die Shift – Taste gedrückt gehalten, was die Tastenkombination Strg + Shift +N ergibt. Die Angabe der Start – Tastenkombination ist optional.
- Eine *Makrobeschreibung* – Information zur Funktion des Makros (optional).
- Die Makrozuordnung in der Dropdown-Liste MAKRO SPEICHERN IN:

  soll ein Makro der Arbeitsmappe in der er aufgezeichnet wurde, zugeordnet werden, wird hier DIESE ARBEITSMAPPE gewählt, die Auswahl PERSÖNLICHE MAKRO-ARBEITSMAPPE macht das Makro allgemeinverfügbar in allen Arbeitsmappen.

Nach dem Betätigen der Schaltfläche OK wird die Aufzeichnung gestartet. Ab dem Start werden alle Aktionen – Tastenanschläge, Menüauswahl, Anklicken eines Symbols vom Makrorekorder aufgezeichnet. In der Programmoberfläche erscheint die mit zwei Schaltflächen versehene Symbolleiste AUFZEICHNUNG:

Über die linke Schaltfläche kann die Aufzeichnung gestoppt werden, die rechte Schaltfläche bestimmt, wie die Positionierung des Zellcursors bei der Aufzeichnung interpretiert wird, z.B. beim Wechsel von der Zelle A2 zur Zelle A4:

- als *absoluter Bezug* – gehe zur Zelle A4 (Schaltfläche deaktiviert)
- als *relativer Bezug* – gehe zur zweiten Zelle rechts von der aktiven (Schaltfläche ist aktiviert).

 Standardmäßig arbeitet der Makrorekorder mit absoluten Bezügen. Damit jedoch das aufzuzeichnende Makro Zeilen an beliebiger Stelle einfügen kann, ist es nötig für die Aufzeichnung die Nutzung der relativen Bezüge zu aktivieren.

Die Aufzeichnung läuft in den folgenden Schritten ab:

1. Damit nach der Ausführung des Makros eine Zelle in der Spalte A aktiv wird, ist eine Zelle dieser Spalte zu aktivieren und anschließend die Funktionskombination EINFÜGEN / ZEILEN aufzurufen.

2. Die Zelle der Spalte E der eingefügten Zeile aktivieren und die Formel (hier am Beispiel der Zeile 5) eingeben:

```
=D5 / (1+C5)
```

3. Die Zelle der Spalte G der eingefügten Zeile aktivieren und die Formel (hier am Beispiel der Zeile 5) eingeben:

```
=F5 / (1+C5)
```

4. Letzte Eingabe mit der ENTER – Taste bestätigen und die Aufzeichnung über die linke Schaltfläche der Symbolleiste AUFZEICHNUNG beenden.

2.1.3 Die Ausführung des Makros

Die in diesem und den folgenden Unterkapiteln beschriebenen Techniken des Makroaufrufs gelten sowohl für aufgezeichnete Makros als auch insbesondere für manuell in der VBA – Entwicklungsumgebung programmierte VBA-Module.

Ein aufgezeichneter Makro kann über die Funktionskombination EXTRAS / MAKRO / MAKROS, die den MAKRO – Dialog öffnet,gestartet werden:

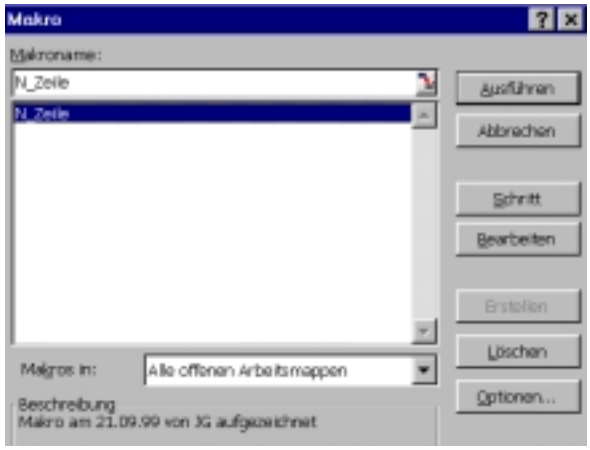


Abbildung 3: Makroaufruf

Das gewünschte Makro ist aus der Liste (hier nur ein Name) zu wählen und über die Schaltfläche AUSFÜHREN zu starten.

Für die richtige Funktion des im Beispiel aufgezeichneten Makros ist es wichtig, daß vor dem Start die Markierung auf einer Zelle der Spalte A steht.

Das Ergebnis:

	A	B	C	D	E	F	G
1	**BelegNr**	**Datum**	**MwSt Satz**	**Einnahme brutto**	**Einnahme netto**	**Ausgabe brutto**	**Ausgabe netto**
2	123	02.05.99	16%	345	297,4138		0
3	195	03.06.99	16%		0	123	106,0345
4	207	05.09.99	8,50%	296	272,8111		0
5							

Abbildung 4: Tabelle nach mehrfacher Ausführung des Beispielmakros

weist noch einige Unzulänglichkeiten insbesondere bezüglich der Formatierungen auf. Diese können vor der Aufzeichnung durchgeführt werden oder während der Aufzeichnung, womit die entsprechenden Anweisungen Bestandteil des Makros werden. Aus Gründen der Übersichtlichkeit wurde hier darauf verzichtet.

Zur Information:

Das aufgezeichnete Makro hat den folgenden Inhalt (Erklärung erfolgt weiter im Text):

```
Sub N_Zeile()
    Selection.EntireRow.Insert
    ActiveCell.Offset(0, 4).Select
    ActiveCell.FormulaR1C1 = "=RC[-1]/(1+RC[-2])"
    ActiveCell.Offset(0, 2).Select
    ActiveCell.FormulaR1C1 = "=RC[-1]/(1+RC[-4])"
    ActiveCell.Offset(1, -2).Select
End Sub
```

2.1.4 Verwaltung von Makros

Über das Dialogfeld aus Abbildung 3 wird die Makroverwaltung geregelt. Über die Schaltflächen ERSTELLEN und BEARBEITEN ist allerdings nicht der Makro – Recorder erreichbar, sondern die VBA – Entwicklungsumgebung, auf die weiter im Text eingegangen wird. In dieser Umgebung werden Makros durch direkte Programmierung erstellt.

Aufgezeichnete Makros können allerdings über BEARBEITEN in der VBA – Umgebung auch nachbearbeitet werden. Durch die Aufzeichnung werden oft Anweisungen in die Makros übernommen, die für die volle Funktionsfähigkeit eines Makros nicht unbedingt benötigt werden und auf diesem Wege entfernt werden können. So sind beispielsweise aus dem obigen Makro einige Teile von Anweisungen gelöscht worden. Man zeichne den Makro testweise auf, um den Unterschied zu sehen.

Über die Schaltfläche LÖSCHEN können nicht benötigte Makros gelöscht werden – dazu Makro in der Liste markieren und die Schaltfläche anklicken.

2.2 Zuordnung zu Tastenkombinationen, Schaltflächen und Menüs

Ein Makro kann über den oben beschriebenen Weg (s. Abbildung 3) aufgerufen werden. Diese Technik des Aufrufs ist jedoch relativ umständlich. Komfortabler ist die Möglichkeit, Makros, später auch VBA – Programme, über Tastenkombinationen, Schaltflächen oder Symbolleisten aufzurufen.

2.2.1 Aufruf über Tastenkombinationen

Sollte vor der Aufzeichnung keine Tastenkombination für dem Aufruf zugeordnet worden sein, kann dieses noch nachträglich geschehen, indem man die Funktionskombination EXTRAS / MAKRO / MAKROS aufruft und im Dialogfeld über die Schaltfläche OPTIONEN die Zuordnung vornimmt.

Viele Tastenkombinationen sind mit EXCEL – Befehlen belegt. Bei der Auswahl sollte eine freie Kombination gewählt werden. In EXCEL belegte Tastenkombinationen können in der Online – Hilfe (hier INDEX) unter dem Begriff TASTENKOMBINATIONEN ermittelt werden.

2.2.2 Aufruf über Schaltflächen

Für den Makroaufruf mittels einer in das Tabellenblatt integrierten Schaltfläche muß das Steuerelement Schaltfläche eingefügt und konfiguriert werden.

Über die Funktionskombination ANSICHT / SYMBOLLEISTEN wird die Symbolleiste FORMULAR

Abbildung 5: Symbolleiste Formular

aktiviert (nicht zu verwechseln mit der Leiste STEUERELEMENTE TOOLBOX, die u.a. optisch zumindest ähnliche Steuerelemente enthält!).

In der eingeblendeten Symbolleiste wird das Steuerelement SCHALTFLÄCHE angeklickt und bei gehaltener linker Maustaste im Tabellenblatt an der gewünschten Position ein Viereck in der Größe der Schaltfläche gezeichnet. Nach dem Loslassen der Maustaste erscheint im Tabellenblatt eine Schaltfläche. Die Größe, Position und Beschriftung können nachträglich noch verändert werden.

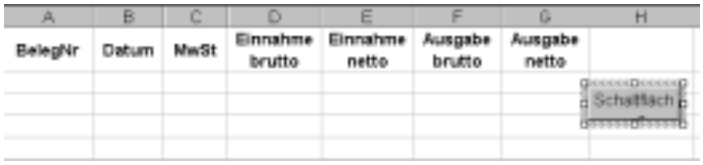


Abbildung 6: Schaltfläche im Tabellenblatt

Mit dem Loslassen der Maustaste wird ebenfalls ein Dialogfenster für die Zuordnung eines Makros zu der eingefügten Schaltfläche eingeblendet:

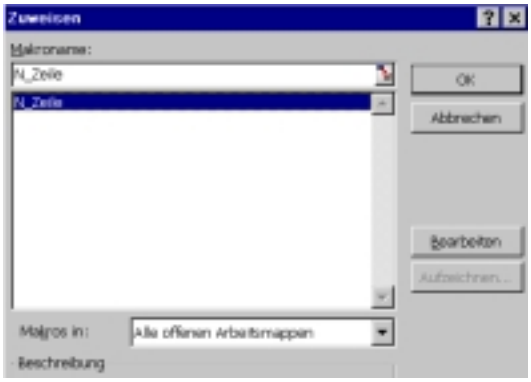


Abbildung 7: Zuordnung eines Makros zur Schaltfläche

Die Zuordnung wird durch die Markierung des gewünschte Makros (hier nur ein Name) und Betätigen der Schaltfläche OK erreicht.

Solange sich die in das Tabellenblatt eingefügte Schaltfläche noch im Bearbeitungsmodus befindet (s. Abbildung 6) kann durch Ziehen mit der Maus ihre Position verändert werden (den Rahmen anklicken, nicht die Schaltfläche !) bzw. über die „Ziehkästchen“ ihre Größe.

Die Beschriftung der Schaltfläche kann beliebig nach dem Markieren des Textes innerhalb der Fläche verändert werden.

Sobald ein Mausklick neben die Schaltfläche (ins Tabellenblatt) erfolgt, ist die Schaltfläche aktiv. Ein Klick darauf startet das zugeordnete Makro.

Bitte beachten: Vor der Ausführung des Beispielmakros muß eine Zelle in der ersten Spalte markiert sein, da sonst keine korrekte Ausführung möglich ist.

✘ Soll die Schaltfläche nach der Aktivierung nachträglich bearbeitet werden, kann es nur im Bearbeitungsmodus geschehen. Dieser ist nur erreichbar, wenn beim Mausklick (linke Maustaste) die STRG-Taste gedrückt wird.

✘ Für die Praxis sollte das Verhalten einer Schaltfläche bezüglich der Änderung von Zellengrößen im Tabellenblatt oder beim Einfügen von Zeilen und Spalten festgelegt werden. Die Schaltfläche kann sich an der Zellengröße, an der Zellenposition oder an beiden orientieren. Dabei ist zu beachten:

- Ein Steuerelement (hier Schaltfläche) sollte möglichst nur eine Zelle überdecken. Überdeckt es mehrere Zellen, reagiert es auf Änderungen der Zellgrößen und Positionen.
- Soll ein über die Grenzen einer Zelle hinausragendes Steuerelement bei Veränderung der Abmessungen der Zelle unverändert bleiben, muß bei aktiviertem Element über die Funktionskombination FORMAT / STEUERELEMENT / EIGENSCHAFTEN die Dialogoption VON ZELLPOSITION UND –GRÖßE UNABHÄNGIG aktiviert werden.
- Soll ein in einer Zelle positioniertes Steuerelement mit dieser Zelle „wandern“ (z.B. beim Einfügen oder Löschen von Zeilen / Spalten), ist in der o.g. Funktionskombination die Option NUR VON ZELLPOSITION ABHÄNGIG zu aktivieren.

2.2.3 Aufruf über Symbolleisten

Für den Aufruf über eine Symbolleiste ist eine Schaltfläche in einer der vorhandenen oder einer neuen Symbolleiste[1] nötig. Die Erstellung und Zuordnung ist in folgenden Schritten möglich:

- Die Funktionskombination EXTRAS / ANPASSEN wählen.
- Im Dialogfenster ANPASSEN die Registerkarte BEFEHLE wählen und darin im Feld KATEGORIEN die Kategorie MAKROS markieren.

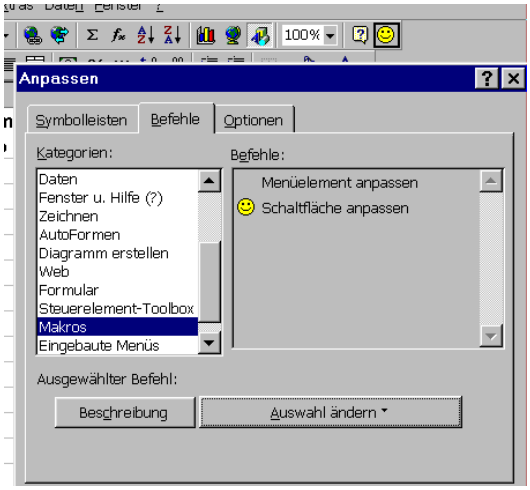


Abbildung 8: Generieren einer Symbolschaltfläche zum Makroaufruf

- Aus der Liste BEFEHLE den Eintrag SCHALTFLÄCHE ANPASSEN bei gedrückter linker Maustaste in eine Symbolleiste ziehen.

[1] Eine neue Symbolleiste wird über die Funktionskombination EXTRAS / ANPASSEN / SYMBOLLEISTEN / NEU erstellt.

- Über die Schaltfläche AUSWAHL ÄNDERN wird ein Kontextmenü aufgerufen, welches eine Makrozuordnung zum Symbol ermöglicht, sowie zusätzliche Definitionspunkte zur Konfiguration des Symbols bietet.

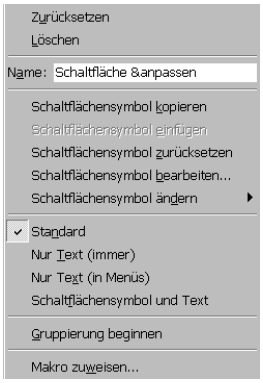


Abbildung 9: Schaltflächen - Definitionsmenü

- Erst nachdem alle Definitionsschritte abgeschlossen sind, wird der Dialog ANPASSEN geschlossen.

Über einen rechten Mausklick auf ein Symbol kann das o.g. Kontextmenü ebenfalls aufgerufen werden, wenn beispielsweise nachträglich die Definition eines Schaltsymbols verändert werden soll.

3 VBA Grundlagen

VBA – *Visual Basic for Applications* ist eine objektorientierte Makroprogrammiersprache mit einer sehr umfangreichen Auswahl von Funktionen und Anweisungen zur Erstellung eigenständiger Programme, die MS Office – Anwendungen automatisieren und oft aus der Sicht des Anwenders komplexe Abläufe in Ihrer Bedienung vereinfachen.

VBA – Module unterscheiden sich je nach Office – Produkt in Ihrer Struktur, hier sollen die Spezifika des Excel – VBA in ihren Grundrissen vorgestellt werden.

VBA – Module sind vom Sprachumfang her in zwei inhaltliche Bereiche teilbar:

- Den die Programmstruktur definierenden *Sprachkern*, der mit dem von VB (Visual Basic) gleichzusetzen ist, mit Schlüsselwörtern, Befehlen, Funktionen und Kontrollstrukturen, die für alle Office – Anwendungen gelten.
- Die anwendungsspezifischen *Objektmodelle*, mit eigenen, den Anwendungen angepaßten OBJEKTEN, EIGENSCHAFTEN, EREIGNISSEN und METHODEN.

Da insbesondere diese vier Begriffe immer wieder als Grundbegriffe der objektorientierten Programmierung im Zusammenhang mit VBA auftreten und somit auch im Inhalt dieser Unterlage öfter vorkommen, sollen sie im folgenden vorgestellt werden.

OBJEKTE stellen einen zentralen Bestandteil fast aller VBA – Anwendungen dar. Es sind bestimmte Teile einer Anwendung – Excel selbst, die Arbeitsmappe, das Tabellenblatt, Zellen (-bereiche).

Objekte besitzen EIGENSCHAFTEN (benannte Attribute), beispielsweise Größe, Farbe, Position, Name usw..

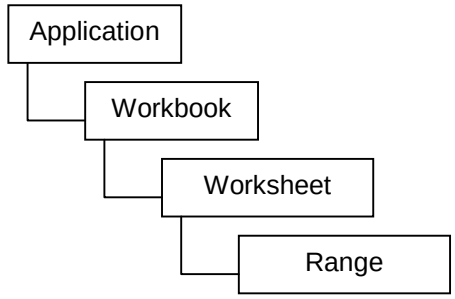
Das Verhalten eines Objekts wird über die METHODEN (= Prozeduren), die auf ein Objekt angewandt werden, bestimmt.

3.1 Objekte und ihre Hierarchie

Excel besteht aus mehr als 200 Objekten, deren Eigenschaften und Methoden frei zugänglich sind. Zu den wichtigsten Objekten zählen:

APPLICATION	Excel selbst / Excel – Fenster
WORKBOOK	Die Excel – Arbeitsmappe
WORKSHEET	Ein Tabellenblatt
RANGE	Zellenbereich , bestehend aus einer oder mehrerer Zellen

Die Objekte stehen in hierarchischer Abhängigkeit zueinander. Objekte höherer Stufe beinhalten Objekte untergeordnet Stufe(n). Für o.g. Objekte gilt die Hierarchieordnung.



Die gesamte Objekthierarchie ist natürlich wesentlich komplizierter und umfangreicher. Die folgende Abbildung vermittelt einen erweiterten Ausschnitt:

Abbildung 10: Objekthierarchie von Excel (Ausschnitt)

Für die Programmierung ist es wichtig zu wissen, daß die Objekthierarchie nicht nur ein willkürliches Ordnungsschema ist. Die Kenntnis der festgelegten Position eines Objekts in der Hierarchie ist nötig, um das Objekt zu referenzieren – darauf zuzugreifen.

Beispiel:

```
Worksheets("WS1999/2000").Range("A7")
```

Eine besondere Form von Objekten bilden die AUFLISTUNGEN (Collections). Es sind Gruppen gleichartiger Objekte, zusammengefaßt in einem Container – der *Auflistung*, z.B. die Auflistung WORKSHEETS, die alle Tabellenblätter einer Arbeitsmappe enthält.

3.2 Eigenschaften

Während ein Objekt weitgehend abgeschlossen ist, d.h. sein innerer Aufbau lässt sich i.d.R. nicht beeinflussen, lassen sich seine EIGENSCHAFTEN per Programmanweisung in vielen Fällen verändern. Die Veränderung der Eigenschaften nimmt oft einen wesentlichen Teil eines Programms ein.

Eigenschaften sind benannte Attribute eines Objekts. Sie bestimmen seine Charakteristika wie Größe, Farbe oder Bildschirmposition, aber auch den Zustand, wie beispielsweise *aktiviert* oder *deaktiviert*.

Es gibt Eigenschaften, die lesbar und veränderbar sind, z.B. *Value* (Wert) oder *Name* (Name), andere lassen sich nur abfragen, aber nicht verändern – es sind sog. *Nur – Lese – Eigenschaften*.

Zu den Eigenschaften, die besonders oft verändert werden, gehören:

CAPTION	Beschriftungen von Objekten
NAME	Die Bezeichnung eines Objekts, unter der es angesprochen (referenziert) wird.
SELECTION	Bezeichnet das markierte Application – Objekt.
VALUE	Wert / Inhalt eines Objekts (Zellinhalt, Textfeld – Eintrag)

Alle Eigenschaften haben immer einen aktuellen Wert. Die Anzahl möglicher Werte ist unterschiedlich groß. So besitzen beispielsweise die Eigenschaften *Color* oder *Value* sehr viele Werte, während andere, beispielsweise *Selected*, nur die Werte *False* oder *True* annehmen.

Beispiel:

```
Worksheets("WS1999/2000").Range("A3:D7").Value = 55
```

Die Anweisung weist allen Zellen des Zellbereichs A3:D7 des Tabellenblatts *WS1999/2000* den Wert *55* zu.

3.3 Methoden

Zu den Objekten gehören neben Eigenschaften auch METHODEN. Über Methoden läßt sich das Verhalten von Objekten steuern / verändern. Eine Methode ist eine Aktion, die eine Operation auf einem Objekt ausführen kann. Zu den am häufigsten benutzten Methoden gehören:

OPEN	öffnet eine Arbeitsmappe
CLOSE	schließt eine Arbeitsmappe (ein Workbook-Objekt) oder Excel (ein Application-Objekt).
CLEAR	löscht einen Zellbereich oder ein Diagramm.
AKTIVATE	aktiviert ein Objekt
SELECT	wählt ein Objekt aus

Für den Einsteiger ist es oft problematisch zwischen Eigenschaften und Methoden zu unterscheiden (insbesondere wenn Methoden die gleichen Namen tragen wie beispielsweise Auflistungen). So sind beispielsweise die beiden Anweisungen:

```
Assistant.Visible = true
Assistant.Move 250,275
```

zumindest optisch sehr ähnlich. Es stellt sich die Frage - sind VISIBLE und MOVE Eigenschaften oder Methoden und wenn nicht welches Element von beiden ist eine Methode und welches ein Eigenschaft.

Die Antwort ist hier relativ einfach: Bei Zuweisungen von Eigenschaften wird das Gleichheitszeichen benutzt, bei Methoden benutzt man (optionale) Parameter ohne Gleichheitszeichen.

Bei der Programmierung in der VBA – Umgebung wird die Entscheidung Eigenschaft / Methode zusätzlich durch spezifische Symbole in entsprechenden Auswahlfenstern erleichtert (s. weiter im Text).

Übrigens: Beide Anweisungen beziehen sich auf den Assistenten „Karlchen Klammer“ – die erste mach ihn sichtbar, die zweite verschiebt Ihn entsprechend den angegebenen Parametern.

3.4 Ereignisse

Für die meisten Objekte sind explizite Ereignisse definiert, denen fest zugeordnete *Ereignisprozeduren* zugehören. Ereignisse können Aufrufe von Menüfunktionen, Anklicken von Schaltern, Symbolen und Tasten aber auch Öffnen von Dokumenten, Berechnungen, Veränderungen von Inhalten usw. Allgemein gesehen - Mausklicks, Tastatureingaben und systeminterne Aktionen lösen Ereignisse aus, auf die über entsprechende Prozeduren reagiert werden muß.

4 Die Entwicklungsumgebung

Excel wird zusammen mit einer kompletten Entwicklungsumgebung ausgeliefert. Die Entwicklungsumgebung ist eine eigenständiges Programm, mit einem eigenen Fenstersystem und eigenen Symbolleisten. Der Aufruf kann auf unterschiedlichen Wegen erfolgen:

- Über die Tastenkombination ALT + F11.
- Über die Funktionskombination EXTRAS / MAKRO / VISUAL BASIC EDITOR (für die Erstellung eines neuen Moduls).
- Über die Funktionskombination EXTRAS MAKRO / MAKROS, markieren eines Makronamens und Betätigen der Schaltfläche BEARBEITEN (für die Bearbeitung eines schon existierenden Makros).
- Über die Schaltfläche VISUAL BASIC EDITOR  der VISUAL BASIC – Symbolleiste (einblenden über ANSICHT / SYMBOLLEISTEN).

Abbildung 11:
Die VB-Symbolleiste

Nach dem Aufruf wird das Fenster der Entwicklungsumgebung geöffnet. Existiert für die geöffnete Arbeitsmappe noch kein Makro, wird das Fenster ohne den Editor geöffnet. Das Editor – Fenster kann der Oberfläche über die Funktionskombination Einfügen / Modul hinzugefügt werden:

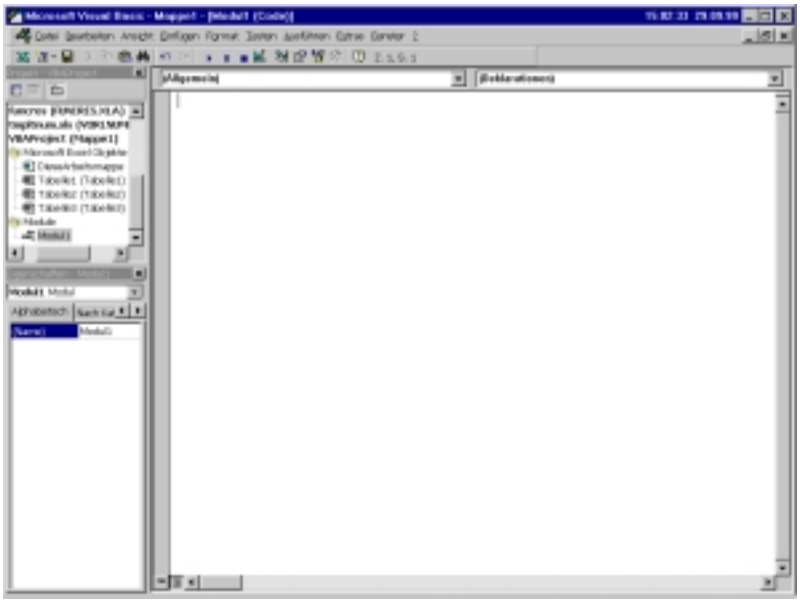


Abbildung 12: Die VBA-Entwicklungsumgebung (hier mit Editor)

➡ Der Wechsel zwischen der Entwicklungsumgebung und der Excel – Oberfläche erfolgt über die Tastenkombination ALT + F11, die Funktionskombination ANSICHT / MICROSOFT EXCEL oder die Excel – Schaltfläche  in der Symbolleiste.

➡ Die Entwicklungsumgebung wird geschlossen über die Systemschaltfläche (oben rechts im Fenster), die Tastenkombination ALT + Q oder die Funktionskombination DATEI / SCHLIEßEN UND ZURÜCK ZU MICROSOFT EXCEL. Wird Excel beendet, so wird auch die Entwicklungsumgebung automatisch geschlossen.

4.1 Der Projekt-Explorer

Der Projekt – Explorer (Projektfenster) ist ein Werkzeug zu Verwaltung von Komponenten gespeicherter VBA - Projekte. Beim Start der Entwicklungsumgebung wird der Explorer standardmäßig geöffnet.

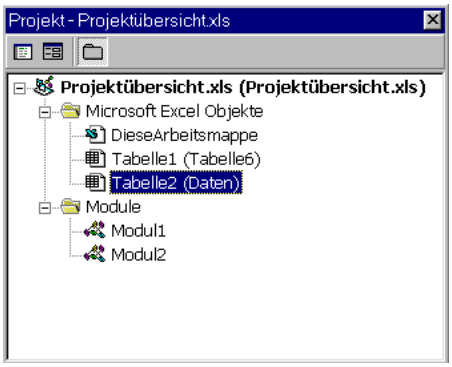


Abbildung 13: Der Projekt - Explorer

Er kann nachträglich über

- die Funktionskombination ANSICHT / PROJEKTEXPLORER,
- die Tastenkombination STRG + R
- das Explorer–Symbol  in der Symbolleiste

geöffnet werden.

Der Projekt - Explorer zeigt alle geöffneten Excel – Arbeitsmappen (PROJEKTE genannt), ihre Tabellenblätter und zu dem Projekten gehörende Module.

In der obigen Abbildung ist das VBA – Projekt *Projektübersicht.xls* geöffnet. Zu diesem Projekt gehört das Excel – Objekt *Arbeitsmappe* mit zwei Tabellen (Tabelle 1 und Tabelle 2) sowie zwei *Module* (Modul 1 und Modul 2)

Über die Symbole der Symbolleiste erhält man Zugang zu den angezeigten Elementen:

 Ist diese Schaltfläche aktiv, werden Tabellenblätter, Module und eventuell vorhandene Dialoge (Formulare) in der Form der Abbildung 13 angezeigt, andernfalls wird nur das Projekt (oberste Zeile in der Abb.) und die Einzelelemente (die innerste Ebene in der Abb.) zur Anzeige gebracht.

 Ein Mausklick auf diese Schaltfläche bewirkt die Anzeige eines markierten Elements (Tabelle, Diagramm, Modul, Formular). Ist beispielsweise eine Tabelle markiert, erfolgt nach dem Mausklick ein Wechsel zu Excel.

 Mit diesem Schalter wird der Editor geöffnet und der dem markierten Element zugeordneter Programmcode geladen. Gleichen Effekt erzielt man durch einen Doppelklick auf ein Element im Explorer – Fenster.

 Das Fenster des Projekt – Explorers wird ähnlich dem des Windows – Explorers bedient – über die Plus- und Minussymbole können die untergeordneten Objekte ein- bzw. ausgeblendet werden.

4.2 Der Objektkatalog

Die hohe Anzahl von Objekten, Eigenschaften, Methoden und Funktionen, die im VBA zur Verfügung stehen, macht es fast unmöglich, den Zweck, die Anwendung und insbesondere auch die Syntax ständig parat zu haben. Hilfe bietet hier der OBJEKTKATALOG.

Der Objektkatalog enthält in einem zweigeteilten Fenster die Auflistung aller verfügbaren Objekte. Im linken Teilfenster (KLASSEN) erscheint eine Liste der Objekte, im rechten (ELEMENTE VON ...) die zu einem markierten Objekt gehörenden Eigenschaften und / oder Methoden:

Abbildung 14: Der Objektkatalog

Das Fenster des Objektkatalogs wird geöffnet über

- die Taste F2
- die Funktionskombination Ansicht / Objektkatalog
- die Schaltfläche Objektkatalog  in der Symbolleiste.

An Symbolen im Fenster des Objektkatalogs ist erkennbar, um welche Elemente es sich handelt:

 Objekt / Klasse

 Modul

 Auflistung

 Eigenschaft

 Ereignis

 Methode / Funktion

 Konstante

Im unteren Bereich des Fensters des Objektkatalogs befindet sich ein Bereich, in dem die Syntax des ausgewählten Objekts, inklusive aller eventueller Parameter angezeigt wird. Zusätzlich wird der Datentyp und Zugehörigkeit zur Objektbibliothek angezeigt – somit ist die Information komplett.

Nach dem Aufruf des Objektkataloges werden i.d.R. die Objekte aller Objektbibliotheken angezeigt. Dadurch ist die Suche wegen der großen Menge von Einträgen etwas umständlich. Sie läßt sich jedoch vereinfachen, wenn man die Anzahl angezeigter Objekte über das im oberen Bereich des Fensters plazierte Dropdown – Feld einschränkt:

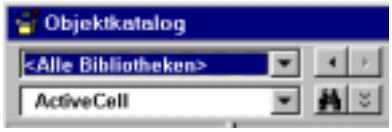


Abbildung 15: Bibliotheksauswahl

Neben der Auswahl ALLE BIBLIOTHEKEN können einschränkend die zur Standardinstallation gehörenden Teilbibliotheken gewählt werden:

EXCEL	Die Excel – Bibliothek mit Eigenschaften, Ereignissen, Methoden und Konstanten zur Manipulation von Excel – Objekten.
MSFORMS	Bibliothek mit Eigenschaften, Methoden und Ereignissen zur Manipulation von Steuerelementen (Schaltflächen, Dropdown-Listen usw.).
VBA	Bibliothek mit allen VBA – Funktionen, Eigenschaften und Konstanten.
VBA-PROJEKT	Bibliothek mit Methoden, Eigenschaften, Funktionen, die zu einer geöffneten Arbeitsmappe bzw. den einzelnen Tabellenblättern dieser Arbeitsmappe gehören. Hierzu gehören auch selbst erstellte Makros und Funktionen. In der Auswahlliste erscheint nicht der Eintrag *VBA-Projekt*, sondern der Name der geöffneten Arbeitsmappe.

Über ein zweites Dropdown-Feld (s. Abbildung 15) kann gezielt nach einzelnen Objekten, Auflistungen, Methoden und Eigenschaften gesucht werden. Der Suchbegriff (auch unvollständige Angabe möglich – s. Abbildung 16) wird in das Eingabefeld der Dropdown – Liste eingegeben und über das *Lupensymbol* die Suche gestartet. Das Suchergebnis wird in einem separaten Teilfenster des Objektkataloges angezeigt:

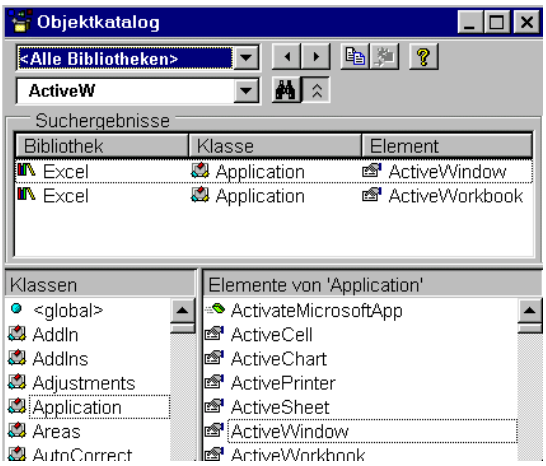


Abbildung 16: Suchergebnis im Objektkatalog

Ein Mausklick auf ein Element des Teilfensters SUCHERGEBNISSE bewirkt die Anzeige der Informationen zur ausgewählten Position im Syntaxbereich des Fensters.

Die rechts neben dem Luppensymbol liegende Schaltfläche blendet die Suchergebnisse wieder aus.

Die Übernahme von Elementen aus dem Fenster des Objektkatalogs in das Editor – Fenster ist relativ umständlich geregelt – sie kann leider nur über die Zwischenablage realisiert werden:

- Ist das gewünschte Element markiert, wird über die Schaltfläche KOPIEREN der Symbolleiste, die Funktionskombination BEARBEITEN / KOPIEREN oder die Tastenkombination STRG + C in die Zwischenablage kopiert.
- Anschließend wird in das Editor – Fenster gewechselt, der Cursor an der Einfügeposition plaziert und über die Schaltfläche EINFÜGEN der Symbolleiste, die Funktionskombination BEARBEITEN / EINFÜGEN oder die Tastenkombination STRG + V eingefügt.

4.3 Der Programmeditor

Die Zentralstellung in der Entwicklungsumgebung nimmt das Fenster des Programmeditors (auch Code – Fenster genannt) ein. Hier wird der Programmcode der Module eingegeben. Der Editor kann nicht direkt aufgerufen werden. Sein Fenster wird automatisch geöffnet, wenn ein Modul geöffnet oder neu erzeugt wird (Funktionskombination EINFÜGEN / MODUL der Entwicklungsumgebung).

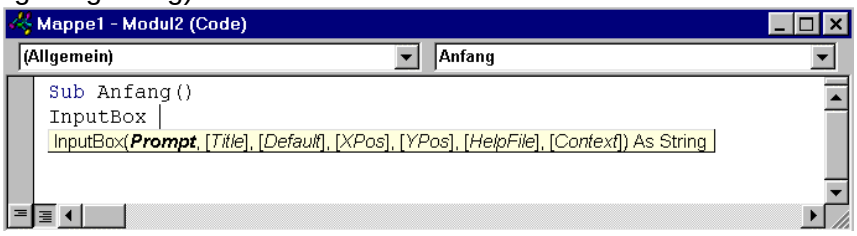


Abbildung 17: Das Fenster des Programmeditors

Das Fenster zeigt immer alle Makros eines Moduls an. Über die rechts oben liegende Dropdown – Liste kann zwischen den einzelnen Makros gewechselt werden. Bei der Eingabe werden Informationen zur Syntax benutzter Funktionen eingeblendet (s.o.).

Bei der Eingabe einer Objektreferenz wird, sobald der Ausdruck mit einem Punkt abgeschlossen wird, eine Auswahlliste der für das Objekt möglichen (verfügbaren) Eigenschaften und Methoden:

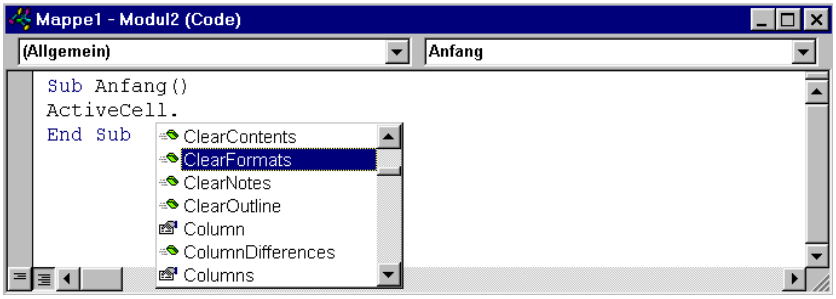


Abbildung 18: Methoden- / Eigenschaften - Auswahl

Per Doppelklick oder Markieren und Betätigen der Tabulatortaste kann die gewünschte Eigenschaft / Methode aus der Liste in den Programmtext übernommen werden.

Der Editor prüft schon während der Eingabe die Syntax. Die Prüfung erfolgt sofort nach Abschluß eines sinnvollen Teilausdrucks und nicht erst nach der Eingabe einer kompletten Anweisung.

✘ Die sofortige Syntaxprüfung kann zu Fehlermeldungen führen, die durch die Unvollständigkeit eines Ausdrucks bedingt sind. Wird die Anweisung jedoch beendet, ist die Fehlermeldung irrelevant und damit verschwunden.

Die Programmzeilen erscheinen im Editor – Fenster in unterschiedlichen Farben:

schwarz	fehlerfreie Programmzeilen, Bezeichner, Objektnamen, Eigenschaften und Methoden.
grün	Kommentarzeilen
blau	VBA – Schlüsselwörter
rot	Compiler – Fehler, auch unvollständige oder fehlerhafte Ausdrücke; nach Korrektur bzw. Ergänzung Wechsel zu *schwarz*.

Weitere Hinweise zum Editor – Fenster siehe Broschüre VBA – GRUNDLAGEN.

4.4 Das Direktfenster

Für das schnelle Testen von Anweisungen kann das DIREKTFENSTER benutzt werden. Über die Funktionskombination ANSICHT / DIREKTFENSTER oder die Tastenkombination STRG + G geöffnet,

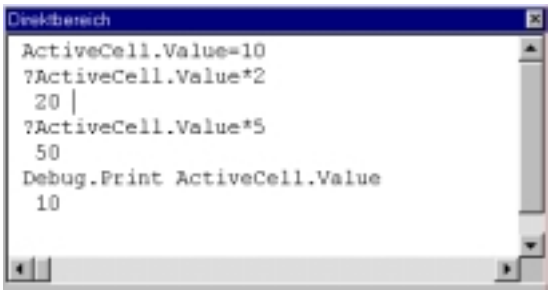


Abbildung 19: Das Direktfenster

ermöglicht es die Kontrolle einzelner Anweisungen.

In der obigen Abbildung wird beispielsweise in der ersten Anweisung der aktiven Zelle eines Tabellenblattes der Wert 10 zugeordnet, die mit einem Fragezeichen **?** beginnenden Zeilen führen Operationen auf der aktiven Zelle aus und geben unmittelbar das Ergebnis in das Direktfenster aus, die Zeile mit DEBUG.PRINT gibt den Wert der aktiven Zelle zur Laufzeit aus.

5 Das Sprachkonzept von VBA

Bevor die VBA – Sprachelemente vorgestellt werden, sollen einige Voraussetzungen für die VBA- Programmierung erläutert werden:

- **Module**
  Ein aufgezeichneter Makro oder ein manuell erstellter Programmcode müssen im Projekt gespeichert werden. Dafür wird ein Art Container benutzt – ein Modul. Erzeugt wird ein Modul über die Funktionskombination EINFÜGEN / MODUL der Entwicklungsumgebung.

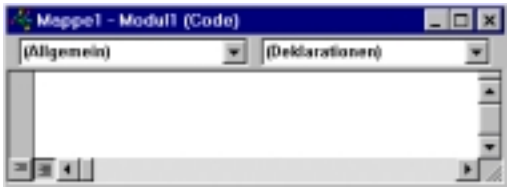


Abbildung 20: Das Modulfenster

  Der erste Aufruf öffnet ein leeres Fenster des Editors mit dem allgemeinen Bereich des Moduls, in dem i.a. Deklarationen von Variablen und Konstanten bzw. Informationen (Anweisungen), die für das gesamte Modul Gültigkeit haben sollen, untergebracht werden (Anfangsbereich oder Deklarationsbereich eines Moduls).

- **Prozeduren**
  Prozeduren sind Gruppen von Anweisungen, als kleinste selbständige Einheiten eines VBA – Programms. Sie haben ein festes Gerüst – sie beginnen mit der Anweisung *Sub* und enden mit *End Sub*:

  Sub Rechnen ()
  .
  Anweisungen
  .
  End Sub

  Der Anweisung *Sub* folgt der PROZEDURNAME. Dieser darf maximal 255 Zeichen lang sein, sollte außer dem Unterstrich keine Sonderzeichen enthalten. Das erste Zeichen muß ein Buchstabe sein, das Leerzeichen und der Bindestrich sind nicht zulässig.

  Zu beachten sind die Argumentklammern – auch wenn keine Argumente übergeben werden, müssen diese Klammern gesetzt werden. Beim Schreiben der Anweisung setzt der Editor die Klammern automatisch.

- **Anweisungen**
  Eine Anweisung ist eine syntaktische Codeeinheit für Definitionen, Deklarationen oder Operationen. Pro Programmzeile wird normalerweise eine Anweisung geschrieben. Soll eine Programmzeile mehrere Anweisungen enthalten, müssen diese durch einen Doppelpunkt (**:**) voneinander getrennt sein.

  Eine Anweisung kann ausführbar oder nicht ausführbar sein. Nicht ausführbare Anweisungen setzen den logischen Programmablauf nicht fort, da sie i.d.R. nur Definitionen bzw. Deklarationen enthalten.

- **Groß- und Kleinschreibung**

  Grundsätzlich wird zwischen Groß- und Kleinschreibung bei Namen von Subs, Funktionen oder Variablen nicht unterschieden. Werden der besseren Lesbarkeit wegen Groß- und Kleinbuchstaben benutzt, so behält VBA die Schreibweise der Namen bei.

- **Argumente in Funktionen und Methoden**

  Werden, anders als in Excel durch Kommata und nicht durch Semikolons voneinander getrennt:

```
MsgBox "Auswertung", vbOKOnly
```

Argumentenklammern sind nur dann erforderlich, wenn ein Rückgabewert auszuwerten ist, d.h. wenn der Term mit der Funktion oder Methode rechts vom Gleichheitszeichen steht:

```
Ausw = MsgBox("Auswertung", vbOKOnly )
```

Argumentenwerte werden mit Hilfe des Operators **:=** und Argumentenbezeichnungen zugewiesen (nicht mit dem normalen Gleichheitszeichen).

```
MsgBox Prompt := "Auswertung", Buttons := vbOKOnly
```

Die Verwendung dieser Methode ist insbesondere dann empfehlenswert, wenn nicht alle Argumente belegt werden sollen. Bei der einfacheren Methode (s. erstes Beispiel) müssen für alle nicht belegten Argumente Kommata als Platzhalter angegeben werden, wenn das / die folgenden Argumente benutzt werden. Dieses entfällt bei der direkten Zuweisung. Auch die Reihenfolge der Argumente ist dann beliebig.

- **Kommentarzeilen**
  Kommentarzeilen werden in VBA – Programmen durch ein Hochkomma als erstes Zeichen der Zeile gekennzeichnet. Gültige Anweisungen können durch Voranstellen eines Hochkommas inaktiviert werden.

```
'Ausgabe einer Meldung
MsgBox Prompt := "Auswertung", Buttons := vbOKOnly
```

  Auch „nachgestellte“ Kommentare sind möglich:

```
MsgBox Prompt := "Auswertung", Buttons := vbOKOnly     'Ausgabe Meldung
```

- **Mehrzeilige Anweisungen**

  können mittels des Unterstrichs _ gebildet werden. Allerdings kann der Umbruch nicht an einer beliebigen Stelle geschehen – nur vor oder hinter Elementen eines Ausdrucks ist er erlaubt. Vor dem Unterstrich muß ein Leerzeichen stehen:

```
MsgBox "Es wurde ein falscher Wert eingegeben", _
vbOKOnly _
"Eingabefehler"
```

  Hinter einem Unterstrich darf kein Kommentar eingefügt werden.

  Bei Fortsetzungen in Zeichenketten müssen diese durch ein Anführungsszeichen abgeschlossen werden und mit dem Verknüpfungsoperator **&** verknüpft werden:

```
MsgBox "Es wurde ein falscher"  & _
"Wert eingegeben", _
vbOKOnly _
"Eingabefehler"
```

5.1 Datentypen

VBA kennt die folgenden Datentypen (Variablentypen):

Typ	Platzbedarf /Bytes)	Wertebereich
Byte	1	0 bis 255
Boolean	2	*True / False*
Currency	8	Festkommazahl mit 15 Stellen vor und 4 Stellen nach dem Komma
Date	8	Datum und Uhrzeit, Datum: 01.01.0100 bis 31.12.9999, Uhrzeit: 00:00 bis 23:59:59
Double	8	Fließkommazahl mit 16 Stellen Genauigkeit
Integer	2	-32.768 bis 32.767 Ganzzahl !
Long	4	-2.147.483.648 bis 2.147.483.647 Ganzzahl !
Object	4	Jeder Objektverweis
Single	4	Fließkommazahl mit 8 Stellen Genauigkeit
String (fest)	Zeichenfolgenlänge	1 bis 65400 Zeichen
String	10 + 2 pro Zeichen	Zeichenzahl ab Vers.4 nur durch RAM beschränkt
Variant (numerisch)	16	
Variant (alphanumerisch)	22 + Textlänge	
Benutzerdefiniert	wie Einzelelemente	wie Einzelelemente

Hinweis:

- Der Datentyp *Variant* ist ein universeller Datentyp. Er gilt als Voreinstellung für alle Variablen, für die kein expliziter Datentyp per Deklaration angegeben wird. Variablen mit dem Datentyp *Variant* passen sich automatisch den in ihnen gespeicherten Daten an.
  Dieser Datentyp ist allerdings nicht optimal handhabbar – er ist sehr speicheraufwendig, Variablen mit diesem Typ können innerhalb des Programms ihren Datentyp beliebig wechseln, was zu unnötigen Fehlermeldungen führen kann, wenn beispielsweise mit Zeichenketten und numerischen Werten gerechnet wird.
  Es wird daher empfohlen, möglichst den Variablen andere, zu den Werten optimal passende Datentypen der Deklaration zuzuweisen (siehe auch weiter im Text).

```
Dim Variable1
   Variable1 = "123"
         .
   Variable1 = Variable1 + 20
         .
   Variable1 = "ABC" & Variable1
```

  Nach der ersten Zuweisung enthält Variable1 die Zeichenkette 123 (ist also alphanumerisch), nach der zweiten den Zahlenwert 143 (numerisch) und nach der dritten wiederum eine Zeichenkette ABC143 (alphanumerisch).

5.2 Operatoren, Operanden, Ausdrücke

Bei der Durchführung von Berechnungen oder der Arbeit mit Zeichenfolgen werden in der Regel Ausdrücke, Operatoren und Operanden verwendet.

5.2.1 Ausdrücke

Ein AUSDRUCK besteht aus Konstanten, Variablen, Funktionen oder anderen Ausdrücken, die mit Hilfe von Operatoren verknüpft sind. Je nach Wert, den ein Ausdruck repräsentiert, spricht man von einem numerischen, alphanumerischen, logischen oder Datumsausdruck. Als Operanden eines Ausdrucks können einfache Werte (Ziffern, Zeichen allgemein), Variable, Konstante, aber auch VBA – Funktionen, Methoden bzw. selbst definierte Funktionen benutzt werden.

5.2.2 Operatoren

Operatoren verknüpfen und manipulieren Variablen, Werte und Ausdrücke. Im VBA existieren vier Kategorien von Operatoren, die sich nach den vier Grunddatentypen orientieren: Zahl, Zeichen(kette), logisch und Datum.
Die Operatoren unterliegen bestimmten Vorrangregeln. Diese bestimmen die Reihenfolge der Ausführung der Operationen, insbesondere wenn im Ausdruck mehrere Operatoren vorkommen.
Generell gilt für die Operatoren die Regel „Punktrechnung vor Strichrechnung“, allerdings ist es kaum möglich, generelle Regeln für die Rangfolge der Operationen festzulegen, insbesondere wenn in Ausdrücken Operatoren verschiedener Kategorien vorkommen.

5.2.2.1 Arithmetische Operatoren

VBA unterscheidet insgesamt sieben arithmetische Operatoren:

Operator	Bedeutung
+	Addition
-	Subtraktion / Negation
*	Multiplikation
/	reguläre Division
\	ganzzahlige Division
^	Potenzierung
Mod	Modulo

Für die arithmetischen Operatoren gelten die üblichen Vorrangregeln (Punkt vor Strich). Andere Reihenfolge der Operationen muß evtl. durch Klammern festgelegt werden.

*5 + 6 * 8*
*(5 + 6) * 8*

Das Ergebnis einer arithmetischen Operation übernimmt den Datentyp des Operanden mit der größten Genauigket (in der Reihenfolge: *Currency, Double, Single, Long, Integer*)

5.2.2.2 Vergleichsoperatoren

Vergleichsoperatoren können auf fast allen Datentypen angewandt werden. Das Ergebnis des Vergleichs ist ein Wahrheitswert - entweder TRUE oder FALSE. Besitzt einer der zu vergleichenden Ausdrücke den Wert NULL ist das Ergebnis des Vergleichs ebenfalls NULL.
➡ Der Wert NULL tritt oft bei Vergleichen von Variablen, von denen eine einen ungültigen Wert enthält, auf.

Vergleichsoperationen können im VBA mit Hilfe folgender Operatoren durchgeführt werden:

Operator	Bedeutung
<	kleiner als
<=	kleiner oder gleich
>	größer als
>=	größer oder gleich
=	gleich
<>	ungleich
Is	Objektvergleich
Like	Mustervergleich

Die zu vergleichenden Ausdrücke müssen vom Typ her verträglich sein.
Zusätzlich gehören in den Bereich der Vergleichsoperatoren noch zwei, in der Tabelle nicht aufgeführte Operatoren:

- IS vergleicht zwei Variablen in Bezug darauf, ob sie auf dasselbe Objekt verweisen:

  Ergebnis = Objekt1 Is Objekt2

Wenn sowohl Objekt1 als auch Objekt2 auf dasselbe Objekt verweisen, ist das Ergebnis TRUE, andernfalls ist das Ergebnis FALSE.

- LIKE dient dem Vergleich einer Zeichenfolge mit einem Muster:

Ergebnis = Zeichenfolge Like Muster

Das Argument MUSTER kann unterschiedliche Zeichen, Listen mit Zeichen, Platzhalter oder Zeichenbereiche enthalten:

```
Test1 = "aBBBa" Like "a*a"    ' Liefert True.
Test1 = "F" Like "[A-Z]"      ' Liefert True.
Test1 = "F" Like "[!A-Z]"     ' Liefert False.
Test1 = "a2a" Like "a#a"      ' Liefert True.
Test1 = "aM5b" Like "a[L-P]#[!c-e]" ' Liefert True.
Test1 = "BAT123khg" Like "B?T*"     ' Liefert True.
Test1 = "CAT123khg" Like "B?T*"     ' Liefert False.
```

Wegen der recht umfangreichen Beschreibung, insbesondere zur Benutzung von Sonder- und Ersatzzeichen, sei hier auf die Online - Hilfe verwiesen.

5.2.2.3 Logische Operatoren

Boolsche Ausdrücke verwenden häufig einen der in VBA bekannten sechs logischen Operatoren, mit deren Hilfe mehrfache Prüfungen von Ausdrücken vorgenommen werden können.
Die allgemeine Syntax von Anweisungen mit logischen Operatoren lautet:

Ergebnis = [Ausdruck1] Operator Ausdruck2

Die Operatoren und Ihre Funktionen:

Operator	Bedeutung	Ergebnis = *True*, wenn
Not	Negation / Nicht	Ausdruck = *False*
And	Konjunktion / Und	beide Ausdrücke = *True*
Or	Disjunktion / inklusives Oder	mindestens ein Ausdruck = *True*
Xor	Exklusion / exklusives Oder	genau ein Ausdruck = *True*
Eqv	Äquivalenz	beide Ausdrücke = *True* oder *False*
Imp	Implikation	außer wenn Ausdruck1=*True* und Ausdruck2=*False*

Der *Not* - Operator wirkt nur auf Operanden, alle anderen logischen Operatoren werden zur Verknüpfung von Ausdrücken verwendet.

```
Zahl1=1, Zahl2=5, Zahl3=10
Erg = Not(Zahl1 < Zahl2)                      ergibt False
Erg = (Zahl2 > Zahl1) And (Zahl3 > Zahl2)     ergibt True
Erg = (Zahl2 < Zahl1) Or (Zahl3 < Zahl2)      ergibt False
Erg = (Zahl2 < Zahl1) Xor (Zahl3 > Zahl2)     ergibt True
Erg = (Zahl2 < Zahl1) Eqv (Zahl3 < Zahl2)     ergibt True
Erg = (Zahl2 > Zahl1) Imp (Zahl3 < Zahl2)     ergibt False
```

In bestimmten Fällen (Behandlung von Tastatur- und Mausereignissen) werden logische Operatoren in Vergleichen von *Bitfeld* - Werten eingesetzt. Solche Vergleiche werden bitweise durchgeführt - verglichen werden Bits, die jeweils an gleichen Positionen in der Bitkette von Ausdrücken stehen. Der NOT - Operator invertiert in solchen Vergleichen jedes Bit eines Operanden (aus 0 wird 1 und umgekehrt).

```
Wert1 = 4                    0000 0100
Wert2 = 7                    0000 0111
Erg = Wert1 And Wert2        0000 0100
```

5.2.2.4 Verkettungsoperatoren

Verkettungsoperatoren werden zur Verknüpfung von Zeichenfolgen verwendet. Verketten lassen sich sowohl Ausdrücke mit alphanumerischen Inhalten als auch gemischte, mit alphanumerischen und numerischen Inhalten.

Für Verkettungen von Zeichenfolgen wird der Operator **&** benutzt.

So wurden beispielsweise in einigen bisherigen Beispielen Verkettungen der Form:

```
Dim Nr As Integer
NR = 23
Ergebnis = "Laufende Nr.: " & NR
```

verwendet, in denen eine *String* - Konstante mit einer numerischen Variablen verkettet wird. Verkettungen dieser Art bleiben durchaus fehlerfrei, da in solchen Fällen Ausdrücke, die keine alphanumerische Kette darstellen, in den Typ *Variant* umgewandelt werden und dem Ergebnis der Verkettung ebenfalls dieser Typ zugewiesen wird.

Ein zweiter möglicher Verkettungsoperator ist das Zeichen „**+**“. Da jedoch das gleiche Zeichen für Additionen benutzt wird und bei Verkettungen nur Verknüpfungen von *String*- und *Variant* - Typen (hier nur mit alphanumerischem Inhalt) fehlerfrei möglich sind, sollte man, um unnötige Fehler zu vermeiden, von der Verwendung dieses Operators bei Verkettungen absehen.

5.3 Variablen, Konstanten, Arrays

5.3.1 Konstanten

Konstante sind feste, vordefinierte Werte (Zahlen oder Zeichenfolgen), die zur Laufzeit nicht geändert werden können. Die Verwendung von Konstanten erlaubt eine einfachere und leichtere Handhabung von Werten (Wert wird nur einmal gesetzt und bei Bedarf nur über den Namen der Konstanten abgerufen) und führt zur besseren Lesbarkeit eines Programms.

Die Deklaration einer Konstanten (im allgemeinen Teil des Moduls, siehe Abbildung 20) wird mit den Schlüsselwort CONST eingeleitet, es folgen der Name und die Wertzuweisung.

Die allgemeine Form

[Public][Private] Const Konstantenname [As Typ] = Ausdruck

PUBLIC für Konstanten, die allen Prozeduren in allen Modulen zur Verfügung stehen sollen.

PRIVATE für Konstanten, die nur in dem Modul verfügbar sein sollen, in dem sie deklariert wurden.

Als TYP sind die zulässigen VBA - Datentypen möglich. Wird As Typ nicht angegeben, paßt sich der Typ automatisch so an, daß er für den angegebenen Ausdruck optimal ist.

```
Const Wert1 = 57

Const Meldung = "Bitte Namen eingeben! "

Private Const Summe As Integer = Zahl1 +  Zahl2
```

Nach der Deklaration können die Konstanten in allen Ausdrücken benutzt werden. Bei Operationen ist jedoch der Typ zu beachten.

5.3.2 Integrierte Konstanten

Systemdefinierte Konstanten (*integrierte Konstanten*) bilden eine Gruppe von Konstanten, die von Steuerelementen und Anwendungen bereitgestellt werden. Der zur Verfügung stehende umfangreiche Vorrat an integrierten Konstanten ist im Objektkatalog von VBA einzusehen.

 Der Präfix des Namens einer integrierten Konstanten deutet auf ihre Herkunft hin. Konstanten mit dem Präfix VB stammen beispielsweise aus der Objekt-Library von VBA, solche mit dem Präfix GRD gehören zum Tabellen-Steuerelement. Gebräuchliche Präfixe des EXCEL – VBA sind:

Präfix	Zugehörigkeit zu
vb	VBA
xl	Excel
Mso	MS Office
fm	MSForms - Bibliothek

5.3.3 Variablen

Variablen sind Platzhalter für Daten (Zahlen, Text), die im Programmverlauf verändert werden können. Jede Variable besitzt einen Namen, mit dem auf sie im Programmverlauf Bezug genommen werden kann und einen Typ (Datentyp), der bestimmt, welche Art Informationen in der Variablen gespeichert werden kann.

Die Variablennamen unterliegen den folgenden Regeln:

- Ein Variablenname beginnt immer mit einem Buchstaben.
- Ein Variablenname kann maximal 255 Zeichen lang sein.
- Ein Variablenname muß innerhalb des Gültigkeitsbereichs eindeutig sein.
- Ein Variablenname darf keinen Punkt und kein Typkennzeichen[2] enthalten.

5.3.3.1 Variablendeklaration

Eine Variable muß vor ihrer Verwendung nicht explizit deklariert werden, weil beim Antreffen eines unbekannten Namens im Programm automatisch eine Variable mit diesem Namen deklariert wird (*implizite Variablendeklaration*).

```
NettoBetrag = 345.53
Vorname = "Johannes"
```

Dies kann jedoch leicht zu Fehlern führen. Erscheint im Programmtext der Variablenname mit einem Schreibfehler, so erfolgen unterschiedliche implizite Deklarationen mit evtl. anschließenden Wertzuweisungen an scheinbar gleiche, in der Praxis jedoch unterschiedliche Variablen (z.B. *Ergebns= ....* statt *Ergebnis=...)*.

Es wird daher empfohlen, eine *explizite Variablendefinition* aller benutzten Variablen im Deklarationsabschnitt eines Moduls mittels der Anweisung

```
Option Explicit
```

vorzunehmen.

 Eine explizite Variablendeklaration kann erzwungen werden, indem auf der Registerkarte EDITOR (erreichbar über die Funktionskombination EXTRAS - OPTIONEN der Menüleiste) die Option VARIABLENDEKLARATION ERFORDERLICH aktiviert wird.
Zur Definition der Variablen wird die Anweisung

```
Dim Variablenname [As Typ]
```

[2] *Typkennzeichen* sind an einen Variablennamen angehängte Sonderzeichen zur Charakterisierung des Datentyps, den die Variable aufnehmen kann. Diese wurden insbesondere in älteren VBA – Versionen benutzt und werden in dieser Unterlage nicht mehr erläutert.

verwendet. Mit der optionalen Angabe AS TYPE kann der Variablentyp festgelegt werden. Wird dieser nicht angegeben, erhält die Variable den Typ VARIANT.

```
Dim Summe As Double
Dim Vorname As String
Dim Wert1 As Variant
```

In einer DIM – Anweisung können auch mehrere Variablen des gleichen Typs oder unterschiedlicher Typen deklariert werden. Dabei sind die Variablennamen bzw. Variablennamen mit der AS – Angabe durch Kommata voneinander zu trennen:

```
Dim Summe, Ergebnis, Zwischen As Double
Dim Vorname As String, Profit As Single, Umsatz As Currency
```

Die Variablentypen entsprechen den weiter oben vorgestellten Datentypen :

Typ	As - Schlüsselwort
Zeichenkette	String
Ganze Zahl (kurz)	Integer
Ganze Zahl (lang)	Long
Fließkommazahl (einfache Genauigkeit)	Single
Fließkommazahl (doppelte Genauigkeit)	Double
Logisch	Boolean
Datum	Date
Alle Typen	Variant

✘ Man sollte Variablen möglichst immer explizit deklarieren. Damit wird die Fehleranfälligkeit der Programme erheblich verringert.

✘ Explizit deklarierte Variablen werden automatisch initialisiert (mit Werten vorbelegt). String – Variablen erhalten als Wert einen Leerstring *""*, numerische den Wert *0*.

5.3.3.2 Geltungsbereich von Variablen

Auf Prozedurebene deklarierte Variablen stehen nur in der jeweiligen Prozedur zur Verfügung. Sie werden LOKALE VARIABLEN genannt.
Lokale Variablen werden entweder mittels der DIM - Anweisung

```
Dim Zahl1 As Integer
```

oder mittels der STATIC - Anweisung

```
Static Zahl1 As Integer
```

deklariert.
Die mittels der STATIC - Anweisung deklarierten Variablen (STATISCHE VARIABLEN) behalten iher Gültigkeit während der gesamten Laufzeit der Anwendung (im Unterschied zu den mittels DIM deklarierten, die nach dem Verlassen der Prozedur ihre Gültigkeit verlieren).
Auf Modulebene (im Deklarationsabschnitt) deklarierte Variablen stehen allen Prozeduren eines Moduls zur Verfügung. Sollen sie auch Prozeduren anderer Module zur Verfügung gestellt werden, müssen sie über

```
Public Variable [As Typ]
```

deklariert werden (öffentliche Variable).
Variable, die nur in einem Modul generell bekannt sein sollen, können auch mit

```
Private Variable [As Typ]
```

deklariert werden. Auf Modulebene ist diese Deklarationsform der Deklaration mit DIM äquivalent, wird aber oft wegen dem Unterschied zu PUBLIC der DIM - Deklaration vorgezogen.

Deklaration	Gültigkeitsbereich
Variable = Wertzuweisung oder *Sub Prozedur()* *Dim Variable* . *End Sub*	lokale Variable, nach Verlassen der Prozedur nicht mehr gültig.
Private Variable As Typ *Sub Prozedur ()* *Variable = Wertzuweisung* *End Sub*	private Variable – in allen Prozeduren und Funktionen eines Moduls gültig.
Public Variable As Typ *Sub Prozedur ()* *Variable = Wertzuweisung* *End Sub*	öffentliche Variable – in allen Prozeduren und Funktionen eines Projekts gültig, solange das zugehörige VBA - Projekt aktiv ist.
Static Variable As Typ	statisch – gültig, solange VBA – Projekt aktiv

5.3.4 Arrays (Datenfelder)

Variable können zu Feldern (Arrays) zusammengefaßt werden. Im Daten - Array existieren mehrere Variablen gleichen Namens, Unterscheidungsmerkmal ist je nach Dimension eines Arrays ein Index oder mehrere Indizes.

Datenfelder können eine feste Größe besitzen oder dynamisch aufgebaut sein.

Deklarationsformen und Gültigkeitsbereiche von Datenfeldern entsprechen den der Variablen, mit dem Unterschied, daß bei Datenfeldern bei der Deklaration die Anzahl Felder (Indexwert für Unter- und Obergrenze) angegeben wird. Wird bei der Indexdefinition kein spezieller Wert angegeben, hat der untere Index den Wert 0.

Der Indexwert muß ganzzahlig sein.

```
Dim DaFeld1(30) As Integer
```

definiert ein Datenfeld mit dem Namen DAFELD, dem Typ INTEGER, bestehend aus 31 Elementen.
Unter- und Obergrenze der Indexwerte lassen sich auch direkt definieren:

```
Dim DaFeld2 (10 TO 30)
```

definiert beispielsweise ein Datenfeld mit den Indexwerte 10 bis 30 (bestehend aus 21 Elementen).

In der Regel besitzen alle Elemente eines Datenfeldes den gleichen Typ. Bei Datenfeldern des Typs Variant können jedoch einzelne Elemente unterschiedliche Datentypen enthalten.

Die Einzelelemente eines Arrays werden über ihren Namen und den Indexwert angesprochen:

```
DaFeld1(1) = 20
DaFeld1(5) = 3

DaFeld2(12) = 23.56
DAFeld2(15) = ”Berlin”
```

Neben eindimensionalen Datenfeldern können auch mehrdimensionale Gebilde definiert werden (theoretisch sind max. 60 Dimensionen möglich):

```
DIM DaFeld (1 TO 11, 1 TO 11) As Integer
```

definiert ein Datenfeld mit 144 (12 * 12) Elementen.

Mehrdimensionale (hier. Zweidimensionale) Arrays eignen sich insbesondere gut für den Datenaustausch mit Excel – Tabellen. Das folgende Beispiel demonstriert einen solchen Austausch:

```
Sub Feld_in_Tabelle ()
Dim Feld(1 To 10, 1 To 10) As Variant
Feld(1, 5) = 125
Feld(1, 7) = 132
Feld(3, 2) = 345
Feld(5, 9) =120
Feld(7, 3) = 654
Worksheets("Tabelle2").Range("A1:J10").Value = Feld
```

Der erste Indexwert entspricht hier der Zeile, der zweite der Spalte der Tabelle. Alle nicht definierten (belegten) Tabellenzellen werden mit Nullwerten belegt, evtl. darin liegende ältere Einträge werden damit gelöscht.

 Der Speicherbedarf steigt bei mehrdimensionalen Feldern rapide an, insbesondere bei Feldern des Typs *Variant*, die sehr speicherintensiv sind. Abhilfe - siehe dynamische Datenfelder.

5.3.5 Dynamische Arrays

Der hohe Speicherbedarf von Datenfeldern macht es zweckmäßig, ihre Größe zur Laufzeit eines Programms je nach Bedarf zu ändern (anzupassen). Um dieses zu erreichen, wird zuerst ein Datenfeld definiert, dessen Dimensionsliste leer ist, z.B.:

```
Dim DaFeld () As Integer
```

Die Anzahl benötigter Elemente wird mit der Anweisung[3]

```
ReDim DaFeld (Anzahl) As Integer
```

oder

```
ReDim DaFeld (n TO m) As Integer
```

festgelegt.

 Die REDIM - Anweisung kann nur auf Prozedurebene verwendet werden (keine Verwendung im Deklarationsteil eines Moduls möglich !).
Die Anweisung kann im Programmablauf beliebig oft verwendet werden und damit beliebig oft die Dimension eines Datenfeldes neu gesetzt werden.

 Bei jeder Verwendung der REDIM - Anweisung werden die Elemente eines Datenfeldes neu initialisiert, wodurch die darin enthaltenen Daten verloren gehen. Der Datenverlust läßt sich vermeiden, wenn die REDIM - Anweisung um das Schlüsselwort Preserve erweitert wird:

```
ReDim Preserve DaFeld (n To m) As Integer
```

PRESERVE bewirkt, daß Elemente im genannten Indexbereich bei der Neuinitialisierung ihre Werte behalten.

```
Sub Dyn_Feld ()
Dim Feld1 () As Variant, Anzahl As Long
Anzahl = InputBox("Bitte Anzahl Elemente angeben")
ReDim Preserve Feld(1 To Anzahl)
Feld(Anzahl) = 99
'Wert ausgeben
MsgBox Feld(Anzahl)
```

[3] Hier am Beispiel des schon weiter oben genannten Feldes *DaFeld*.

5.3.6 Benutzerdefinierte Datentypen

Eine Variable, die unterschiedliche, „thematisch zusammengehörende“ und als ein Informationsblock zu behandelnde Informationen aufnehmen soll, wird deklariert mit einem benutzerdefinierten Datentyp.

Variable dieses Typs werden im Deklarationsabschnitt eines Moduls mit Hilfe der TYPE - Anweisung erstellt und können mittels der Schlüsselwörter PRIVATE oder PUBLIC bezüglich ihres Geltungsbereichs festgelegt werden.

Benutzerdefinierte Datentypen können neben unterschiedlichen einzelnen Typen von Daten auch Datenfelder oder andere benutzerdefinierte Datentypen aufnehmen.

Sollen beispielsweise Angaben zur Person gespeichert werden, bietet sich dafür ein benutzerdefinierter Datentyp mit dem Namen PERSANGABEN in der Form:

```
Public Type PersAngaben
  LfdNr As Integer
  Vorname As String * 15
  Name As String * 20
  Titel As String * 10
  Männlich As Boolean
  KfzKz As String * 3
  PLZ As String * 5
  Strasse As String * 20
  Ort As String * 20
  GebDatum As Date
  Kinder As Integer
  Verdienst As Currency
End Type
```

Auf der Basis eines so gebildeten benutzerdefinierten Datentyps müssen im Programm Variablen definiert werden, deren Struktur gleich diesem Datentyp ist, beispielsweise:

```
Dim Klubmitglied As PersAngaben
Dim Senioren As Persangaben
```

Auf die einzelnen Elemente dieses Datentyps wird zugegriffen wie auf Objekteigenschaften:

```
PersAngaben.PLZ =“12345“
Land = PersAngaben.KfzKz
```

Nach der Zuweisung

```
Klubmitglied = PersAngaben
```

erhalten alle Elemente von KLUBMITGLIED die gleichen Werte wie die von PERSANGABEN.

5.3.7 Objektvariablen

Objektvariablen sind Variablen, die auf ein Objekt verweisen (Zellbereich, Tabelle usw.). Sie können ebenfalls mittels der DIM – Anweisung und der Zuweisung des Typs vereinbart werden.

```
Dim Objektvar As Objekttyp
```

Diese Variablen machen den Zugriff auch Objekte einfacher und schneller, weil sich der sonst für den Verweis auf Objekte benötigte Aufbau von umfangreichen Objektreferenzen erübrigt.

Die Wertzuweisung unterscheidet sich aller von der Zuweisung an „normale“ Variablen – sie muß mittels der SET – Anweisung erfolgen:

```
Set Objektvar = Objektausdruck
```

Die Vorteile dieses Variablentyps werden deutlich, wenn innerhalb des Programms mehrfach auf das gleiche Element zugegriffen werden muß:

```
Sub Stati ()

Dim Vollzeit97 As Range
Dim Vollzeit98 As Range
Dim VollzeitG As Range

Set Vollzeit97 = Worksheets("Stud97").Range("B2")
Set Vollzeit98 = Worksheets("Stud98").Range("B2")
Set VollzeitG = Worksheets("Stud99").Range("B2")

Vollzeit97 = 8975
Vollzeit98 = Vollzeit97 * 0.17
VollzeitG = Vollzeit97 + Vollzeit98 * 0.03

End Sub
```

Der Vorteil dieses Verfahrens wird sichtbar, wenn beispielsweise die letzte Zeile aufgelöst wird in eine Anweisung mit normalen Referenzen. Aus

```
VollzeitG = Vollzeit97 + Vollzeit98 * 0.03
```

wird dann eine lange und relativ unübersichtliche Zeile (hier auf mehrere Zeilen verteilt):

```
Worksheets("Stud99").Range("B2") = _
Worksheets("Stud97").Range("B2") + _
Worksheets("Stud98").Range("B2") * 0.03
```

5.4 Kontrollstrukturen

Kontrollstrukturen werden in der Programmierung benutzt, um die physikalische Reihenfolge der Anweisungen im Programmablauf zu verändern. Zu Kontrollstrukturen gehören:

- *Entscheidungsstrukturen*, in denen ein Programm zu einem von mehreren alternativen Anweisungsblöcken verzweigen kann.
- *Schleifenstrukturen*, die die wiederholte Ausführung von Anweisungsgruppen erlauben.

5.4.1 Entscheidungsstrukturen (Verzweigungen)

Entscheidungen erlauben es, in Abhängigkeit von einer Bedingung, unterschiedliche Programmteile auszuführen. Das Programm verzweigt in mehrere Äste, von denen allerdings immer nur einer ausgeführt werden kann.

5.4.1.1 Die If...Then ...Else – Anweisung

Die einfachste Form dieser Anweisung ist die einzeilige If - Anweisung in der Form:

If Bedingung Then Anweisungen1 [Else Anweisungen2]

BEDINGUNG ist ein Ausdruck, dessen Auswertung die Werte TRUE oder FALSE liefert. Ist das Ergebnis TRUE, wird der THEN - Zweig ausgeführt (ANWEISUNGEN1), andernfalls der ELSE - Zweig (ANWEISUNGEN2). Beide Anweisungsteile können aus Einzelanweisungen oder mehreren Anweisungen bestehen. Bestehen sie aus mehr als einer Anweisung, müssen die Einzelanweisungen voneinander durch Doppelpunkte (:) getrennt sein und unbedingt in einer Programmzeile untergebracht werden können:

```
If Zahl1 > 5 Then Meldung = "über 5"
If Zahl1 > 5 Then Meldung = "über 5" Else Meldung = "unter 5"
If Zahl1 < 5 Then Erg1 = Zahl1 + 1 : Erg2 = Zahl1 - 3 : Erg3 =
Zahl1 Mod 2
```

Die gleiche Anweisung kann auch in der Blockform angewandt werden:

```
If A1 = 1 Then

...diese Anweisungen werden nur ausgeführt,
...wenn A1 = 1 ist

EndIf
```

oder

```
If A1 = 1 Then

...diese Anweisungen werden nur ausgeführt,
...wenn A1 gleich 1 ist

Else

...diese Anweisungen werden nur ausgeführt,
...wenn A1 ungleich 1 ist

EndIf
```

Eine erweiterte Form dieser Blockanweisung hat die Syntax:

```
If Bedingung1 Then
... [Anweisungsblock1]
      ... [ElseIf Bedingung2 Then
            Anweisungsblock2]] ...
             .
          [Else
            [Anweisungsblock n]]
EndIf
```

Hier werden die Bedingungen hinter IF oder ELSEIF von oben beginnend solange geprüft und eventuell der dazugehörige Anweisungsblock solange übersprungen, bis eine Bedingung den Wert TRUE liefert. In solchen Fällen wird der entsprechende Anweisungsblock ausgeführt und anschließend zur ersten Anweisung hinter ENDIF gesprungen. Trifft keine Bedingung zu, werden die Anweisungen im ELSE - Zweig ausgeführt und die Programmausführung hinter ENDIF fortgesetzt.

 Treffen Bedingungen in mehreren ELSE - Zweigen zu, wird nur der Anweisungsblock ausgeführt, der auf die erste zutreffende Bedingung folgt und anschließend wird der Programmablauf hinter EndIf fortgesetzt.

```
Sub Vergleich ()

Dim Zahl As Integer
Zahl = InputBox("Geben Sie eine Zahl ein :")

If Zahl > 0 Then
      MsgBox("Die Zahl ist positiv")
Else
      MsgBox(("Die Zahl ist nicht positiv oder Null")
End If

End Sub
```

```
Sub Vergleich2 ()
   Dim Betrag As Integer, Wert As Integer
   Dim Eingang
   Betrag = 450
   Eingang = False
```

```
If Betrag > Wert And Eingang Then
   Meldung = "überzahlt"
ElseIf Betrag = Wert And Eingang Then
   Meldung = "ausgeglichent"
Else
   Meldung = "mahnen !"
End If

End Sub
```

5.4.1.2 Die Iif – Anweisung

Als Alternative zur einfachen If – Anweisung, insbesondere, wenn diese für bedingte Zuweisungen von Werten an Variablen benutzt werden soll, kann die IIF – Funktion dienen. Die Syntax dieser Funktion:

```
Iif(Bedingung, Ausdruck1, Ausdruck2)
```

zeigt die Verwandtschaft mit der WENN – Funktion von Excel. Ähnlich funktioniert auch diese Funktion – AUSDRUCK1 wird zurückgegeben, wenn die Bedingung den Wahrheitswert TRUE liefert, AUSDRUCK2, wenn der Wahrheitswert FALSE ist.

Die Ausdrücke 1 und 2 können Konstanten, Variablen, Funktionen oder Kombinationen dieser Elemente sein, Anweisungen kann IIF jedoch nicht ausführen.

 Statt

```
If Wert > 0 Then
   Aus = "bezahlt"
Else
   Aus = "offen"
End If
```

kann mittels IIF verkürzt werden zu:.

```
Aus = IIf(Wert > 0, "bezahlt", "offen")
```

5.4.1.3 Die Select Case - Anweisung

Eine Alternative zur Blockform der If - Anweisung bildet die SELECT CASE -Anweisung. Ihr Vorteil gegenüber der IF - Anweisung liegt darin, daß sie in einfacherer und übersichtlicherer Form die Definition von Anweisungsgruppen erlaubt. Intern wird sie vom System in einen effizienteren Code umgesetzt und beschleunigt dadurch die Programmausführung.

Ihre Syntax lautet:

```
Select Case Testausdruck
  [Case Werteliste1
   ..[Anweisungen1]]
  [Case Werteliste2
   ..[Anweisungen2]]...
          .
  [Case Else
   ..[Anweisungen n]]
End Select
```

TESTAUSDRUCK ist die auszuwertende ist ein Ausdruck, der einen numerischen Wert oder eine Zeichenkette liefert.

WERTELISTEN können aus einzelnen Werten bestehen, oder aus Ausdrücken der Form:

Ausdruck1, Ausdruck2 [...,Ausdruck n]	Liste von Werten, durch Kommata getrennt
Ausdruck1 To Ausdruck n	Werte mit Unter- und Obergrenze
Is Vergleichsoperator Ausdruck	Wertebereich

Anwendungsbeispiel:

```
Select Case Zahl1
Case Is <= 0
    Meldung = "Zahl ist zu klein"
Case 1, 2, 3
    Meldung = "Zahl ist kleiner 4"
Case 4 To 7
    Meldung = "Zahl ist zwischen 4 und 7"
Case Is > 8 And Zahl1 < 11
    Meldung = "Zahl ist 9 oder 10"
Case Else
    Meldung = "Zahl außerhalb des Bereichs"
End Select
```

```
Select Case Wohnort
Case "Hamburg", "Bremen", "Kiel"
    Meldung = "Norden"
Case "München", "Passau"
    Meldung = "Süden"
Case Else
    Meldung = "leicht erreichbar"
End Select
```

 Das folgende (lustige) Beispiel einer Gewinnauswertung mittels SELECT CASE arbeitet u.a. mit schon weiter oben im Text vorgestellten Objektvariablen:

```
Sub GewinnAuswertung ()
  Dim Umsatz As Range
  Dim Kosten As Range
  Dim Gewinn As Single

  Set Umsatz = Worksheets("Endwerte").Range("D25")
  Set Kosten = Worksheets("Endwerte").Range("G25")
  Gewinn = Umsatz.Value – Kosten.Value

Select Case Gewinn
Case Is < 10
    Meldung = "Werbung !"
Case 11 To 8000
    Meldung = "passabel"
Case 8001 To 15000
    Meldung = "super"
Case Is > 15000
    Meldung = "Wahnsinn"
Case Else
    Meldung = "Manipulation ??"
End Select
End Sub
```

5.4.1.4 Die Switch - Funktion

Die Funktion SWITCH, mit der Syntax:

```
Switch (Bedingung1, Ausdruck1 [, Bedingung2, Ausdruck2]....)
```

vergleichbar mit der SELECT CASE - Struktur:

```
FamStand = Switch(Kenn = 1, "ledig", Kenn = 2 , "verheiratet",
Kenn = 3, "verwitwet")
```

wertet eine Liste von Bedingungen aus und verarbeitet den Ausdruck der zutreffenden Bedingung.

5.4.1.5 Die Choose - Funktion

Die Funktion CHOOSE, mit der Syntax:

```
Choose(Index, Ausdruck1 [, Ausdruck2]...)
```

ebenfalls mit der SELECT CASE - Konstruktion vergleichbar:

```
PersGrösse = Choose(GRIndex, "klein", "mittel", "gross",
"riesig")
```

gibt in Abhängigkeit von Wert eines ganzzahligen Indexes den Wert aus einer Liste der Alternativen aus, dessen Listenposition dem Indexwert entspricht.

5.4.2 Schleifenstrukturen

Mit Hilfe von Schleifenstrukturen lassen sich bestimmte Anweisungsfolgen mehrmals, abhängig von einer bestimmten Bedingung ausführen. Die Anzahl der Schleifendurchläufe wird durch den Wahrheitswert eines Ausdrucks - der SCHLEIFENBEDINGUNG - oder durch den Wert eines numerischen Ausdrucks - des SCHLEIFENZÄHLERS - definieren.

5.4.2.1 While – Schleife

Auf der WHILE – Schleife sind die meisten Schleifenstrukturen aufgebaut. In dieser Schleife wird am Anfang eine Bedingung geprüft und der Schleifenkörper nur dann durchlaufen, wenn die Prüfung den Wert TRUE liefert:

```
While Bedingung
.
Anweisungen
.
Wend
```

Liefert die Prüfung den Wert FALSE, wird die Schleife verlassen und das Programm mit den Anweisungen nach WEND fortgesetzt.

✘ Man sollte bei WHILE – Schleifen unbedingt darauf achten, daß innerhalb der Schleife eine Möglichkeit besteht, die Bedingung auf den Wert FALSE laufen zu lassen, da sonst eine solche Schleife leicht zu einer Endlosschleife werden kann und das Programm dann nur noch über die Tastenkombination STRG + ALT + ENTF abgebrochen werden kann !

```
Sub Versuch ()
Dim Umsatz(1 To 20)
  Wert = 1
  While Wert < 20
     Wert = Wert + 1
     Umsatz(Wert) = 5
  Wend
End Sub
```

5.4.2.2 Do...Loop – Anweisung

Die DO...LOOP – Anweisung wird verwendet, um Anweisungen auszuführen, solange oder bis bestimmte Bedingungen erfüllt sind. Sie ist sicherer als die einfache While – Wend – Anweisung, weil sie beispielsweise über Exit Do (s. weiter im Text) das vorzeitige Verlassen der Schleife ermöglicht.

In der Form:

```
Do [While Bedingung]
.
Anweisung(en)
[Exit Do]
Anweisung(en)
.
Loop [While Bedingung]
```

werden die Anweisungen ausgeführt, solange Bedingung erfüllt ist (hier Zahlen 1 bis 1000):

```
Zähler = 1
Do While Zähler < 1001
     Zähler = Zähler + 2
Loop
```

In der Form:

```
Do [Until Bedingung]
.
Anweisung(en)
[Exit Do]
Anweisung(en)
.
Loop [Until Bedingung]
```

werden die Anweisungen ausgeführt bis die Bedingung erfüllt wird:

```
Do Until Zähler = 1000
     Zähler = Zähler + 2
Loop
```

Die Bedingung kann sowohl am Anfang (bei DO) oder am Ende der Schleife (bei LOOP) stehen, an beiden Stellen gleichzeitig ist sie unzulässig.

Steht die Bedingung im Schleifenkopf, kann der Schleifendurchlauf schon vor dem Eintritt in die Schleife verhindert werden.

Solen die Anweisungen in der Schleife mindestens einmal durchlaufen werden, wird die „Fußvariante“ der DO...LOOP - Anweisung benutzt:

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

In der Anweisungsfolge:

```
Zähler = 2500
Do While Zähler < 1001
   Zähler = Zähler + 1
   Anw.Text = Zähler
Loop
```

werden die Anweisungen der Do - Schleife nicht ausgeführt.

In der Form:

```
Zähler = 2500
Do
   Zähler = Zähler + 1
   Anw.Text = Zähler
Loop While Zähler < 1001
```

werden sie einmal ausgeführt, anschließend die Bedingung geprüft und die Schleife verlassen.

Die Anweisungsfolge:

```
Zähler = 80
Do
Zähler = Zähler + 1
Anw.Text = Zähler
Loop While Zähler < 301
```

zählt beispielsweise von 81 bis 301.

 Die Anweisungsfolge:

```
Zhl = 1
Vergl = ActiveSheet.Range("M25").Value
Do
  Zhl = Zhl + 1
  ActiveSheet.Range("A3").Value = Zhl
    If Zhl = Vergl Then
     Exit Do
    End If
Loop While Zhl < 301
```

bricht die Schleife ab, sobald der Wert von ZHL gleich dem Wert in der Zelle M25 ist..

5.4.2.3 For ... Next – Anweisung

Die FOR ... NEXT - Schleife legt die Anzahl der Wiederholungen des Anweisungsblocks über eine Zählervariable fest. Die Bedingung wird im Kopf der Schleife definiert Der Zähler muß nicht, wie in den bisher vorgestellten Schleifenformen, innerhalb der Schleife verändert werden.

Die allgemeine Form des Anweisung lautet:

```
For Zähler = Anfangswert To Endwert [ Step Schrittweite]

Anweisung(en)
[Exit Do]
Anweisung(en)
.
Next [Zähler]
```

Die numerische Variable ZÄHLER4 wird bei jedem Schleifendurchlauf um den Wert von Schrittweite inkrementiert oder dekrementiert (je nach Vorzeichen von Schrittweite).

Vor dem ersten Schleifendurchlauf wird der Wert von Zähler auf Anfangswert gesetzt und die Anweisungen solange wiederholt, bis der Wert von ZÄHLER größer (positive Schrittweite) oder kleiner (negative Schrittweite) als ENDWERT ist.

 Wird SCHRITTWEITE nicht angegeben, gilt SCHRITTWEITE = 1.

 Der Wert von ZÄHLER sollte innerhalb der Schleife der FOR ... NEXT - Anweisung nicht verändert werden, weil es i.d.R. zu unnötigen Fehlern führt.

```
Dim N As Integer
 For N = 1 To 21 Step 2
     FeldA.AddItem = N ^ 2
 Next
```

berechnet die Quadrate der Zahlen 1, 3, 5, 7, ..., 19, 21.

[4] Ist numerischauch, kann Element eines benutzerdefinierten Datentyps sein.

5.4.2.4 For Each ... Next – Anweisung

Während bei der FOR ... NEXT - Schleife die Anzahl der Durchläufe explizit angegeben wird, legt VBA in der FOR .. EACH - Anweisung diese Zahl selbst implizit fest.

Diese Schleife wiederholt Anweisungen für alle Elemente eines Datenfeldes, wobei die Anzahl der Durchläufe gleich der Anzahl der Elemente ist. Die Schleife ist speziell für Arrays und Auflistungsobjekte geeignet

Die Syntax der Anweisung lautet:

```
For Each Element In Gruppe

Anweisung(en)
[Exit Do]
Anweisung(en)
.
Next [Element]
```

ELEMENT als Argument entspricht der Zählvariablen der üblichen FOR – Schleifen.

GRUPPE bezeichnet eine Auflistung oder ein Array.

 Das folgende Beispiel benennt Tabellenblätter um:

```
Sub Benennen ()
  Nr = 1
  For Each Blatt In Worksheets
   Blatt.Name = "Teil " & Nr
   Nr = Nr + 1
   If Nr = 6 Then
    Exit For
   End If
  Next
End Sub
```

Umbenannt werden die ersten 5 Blätter der aktiven Arbeitsmappe (*Exit For* bei 6). Enthält die Arbeitsmappe mehr als 5 Blätter, bleiben die Benennungen der anderen unverändert.

```
Sub Steuer ()
  Dim Zahlenfeld(5) As Variant
  Dim MwSt As Single
  Feld(0) = 50
  Feld(1) = "Preis"
  Feld(2) = "Rabatt"
  Feld(3) = 120
  Feld(4) = 30
  MwSt = 1.16

  For Each Inhalt In Zahlenfeld
    If IsNumeric(Inhalt) Then
      Inhalt = Inhalt * MwSt
    End If
  Next
End Sub
```

Das obige Beispiel verändert die Elemente des Arrays ZAHLENFELD, allerdings nur dann, wenn der Inhalt eines Elements numerisch ist.

5.4.2.5 Die With – Anweisung

Mit Hilfe der WITH - Anweisung, die als eine Schleife mit nur einem Durchlauf verstanden werden kann, können mehrere Eigenschaften eines Objekts manipuliert werden, ohne daß jedesmal der Objektname angegeben werden muß.

Ihre Syntax:

```
With Objekt
 .. [Anweisungen..]
End With
```

So können beispielsweise die Anweisungen

```
UserForm2.CommandButton.Caption = "OK"
UserForm2.CommandButton.Enabled = True
UserForm2.CommandButton.Width = 100
UserForm2.CommandButton.Height = 30
```

in unterschiedlichen Versionen mittels der WITH – Anweisung umgesetzt werden:

```
With UserForm2
 .CommandButton.Caption = "OK"
 .CommandButton.Enabled = True
 .CommandButton.Width = 100
 .CommandButton.Height = 30
End With
```

```
With UserForm2.CommandButton
.Caption = "OK"
.Enabled = True
.Width = 100
.Height = 30
End With
```

WITH – Anweisungen können ineinander geschachtelt werden:

```
With ActiveCell
  With .Borders
    .LineStyle = xlSingle
    .ColorIndex = 5
  End With
  With .Font
    .Bold = True
    .ColorIndex = 5
  End With
End With
```

Diese Anweisung formatiert den Rahmen und die Zeichen einer aktiven Zelle.

 Bei der Benutzung der WITH – Funktion ist unbedingt auf die korrekte Syntax der Objekt- und Eigenschaftsnamen (Punkte !!) zu achten !

Die folgende WITH - Anweisung setzt mehrere Eigenschaften einer Befehlsschaltfläche mit dem Namen EINFÜGEN:

```
With Einfügen
   .Heigth = 300
   .Width = 900
   .Left = 280
   .Top = 800
   .MultiLine = True
   .Caption = "Einfügen eines Wertes"
End With
```

5.4.2.6 Die Exit For- und Exit Do – Anweisung

Mit Hilfe dieser beiden Anweisungen können die DO- bzw. FOR - Schleifen beim Eintreten einer bestimmten Bedingung vorzeitig verlassen werden.

```
For N = 1 To 50
  Erg.Text = Str(N)
  Zahl1 = N ^2
  If Zahl1 > VerglWert Then
   Exit For
  End If
  Zahl2 = N
Next
```

```
N = 1
Do While N < 51
  Erg.Text = Str(N)
  Zahl1 = N ^2
   If Zahl1 > VergWert Then
    Exit Do
   End If
  Zahl2 = N
N = N + 1
Loop
```

5.5 Konvertierung und Manipulation von Daten.

Für die richtige Funktionsweise von Programmen ist es sehr wichtig zu wissen, welchen Datentyp die verwendeten Variablen zur Laufzeit aufweisen. Fehlerhafte bzw. unterlassene Erkennung und Konvertierung sind die häufigsten Ursachen von Laufzeitfehlern.

Insbesondere der Datentyp VARIANT ist dabei zu beachten, da er praktisch alle anderen Datentypen vertreten kann.

5.5.1 Ermittlung des Datentyps

Der Ermittlung des Datentyps dient im VBA die Funktion VARTYPE(). Sie ermittelt nicht nur den Datentyp einer Variablen, sondern auch den Typ von Ausdrücken, die auch Range – Objekte enthalten dürfen. Die Funktion gibt, je nach Datentyp, die folgenden Werte zurück:

Datentyp	Rückgabewert
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date	7
String	8
Object	9
Error	10
Boolean	11

 Wird die Funktion auf Zellinhalte angewandt, so wird bei numerischen Inhalten keine Differenzierung zwischen LONG, DOUBLE und INTEGER getroffen – alle numerischen Werte werden als DOUBLE – Werte identifiziert.

 Der Variablentyp VARIANT kann Datentypen enthalten, die keinen Bezug zu numerischen oder alphanumerischen Daten besitzen – eine nicht initialisierte VARIANT – Variable hat den Datentyp EMPTY, eine VARIANT – Variable mit ungültigen Daten den Typ NULL.

```
Dim Wert As Range
Set Wert = ActiveSheet.Range("C7")
Inhalt = VaType(Wert)
```

5.5.2 Konvertieren von Datentypen

Für die Konvertierung von Datentypen steht im VBA eine Reihe von Konvertierungsfunktionen zur Verfügung. Die folgende Tabelle enthält eine Auswahl der am häufigsten verwendeten:

Funktion	Konvertierung
Str	numerischer Ausdruck ➔ ein String
Val	Zeichenfolge ➔ numerischer Wert
CStr	Ausdruck ➔ Typ STRING
Cvar	Ausdruck ➔ Typ VARIANT
CInt	Ausdruck ➔ Typ INTEGER
CSng	Ausdruck ➔ Typ SINGLE
CLng	Ausdruck ➔ Typ LONG
CDbl	Ausdruck ➔ Typ DOUBLE
CCur	Ausdruck ➔ Typ CURRENCY
CDate	Ausdruck ➔ Typ DATE
CBool	Ausdruck ➔ Wahrheitswert

Konvertierung numerisch ➔ String

Die Konvertierung numerischer Wert in Zeichenfolgen ist mittels der Funktionen STR bzw. CSTR möglich. Dabei ist folgendes zu beachten:

- STR konvertiert numerische Werte in Zeichenfolgen. Der ausgegebene Wert hat den Typ VARIANT. Bei positiven numerischen Ausdrücken wird ein führendes Leerzeichen, bei negativen ein Minuszeichen ausgegeben.
- CSTR verarbeitet jeden gültigen Ausdruck und konvertiert ihn zum Typ STRING. Sie erzeugt keine führenden Leerzeichen bei positiven Werten.

Konvertierung Zeichenausdruck ➔ numerisch

Für den umgekehrten Weg – die Konvertierung von Zeichenausdrücken in numerische Wert steht die Funktion VAL zur Verfügung:

Enthält der Zeichenausdruck neben Ziffern noch andere Zeichen, werden die Ziffern herausgefiltert. Sind Ziffern allerdings Bestandteil einer Zeichenkette oder stehen sie am Ende einer Zeichenkette, so werden sie nicht erkannt.

Wird innerhalb des zu konvertierenden Ausdrucks keine Ziffer gefunden, gibt die Funktion den Wert 0 zurück.

`Val("abc")` ergibt 0
`Val("1.5abc")` ergibt 1.2
`Val("1,5")` ergibt 1

5.5.3 Manipulieren von Daten

Vergleich von Zeichenfolgen

Der Vergleich von Zeichenfolgen auf Identität kann auf der gesamten Zeichenfolge oder auf Teilstrings erfolgen. Dabei ist jedoch zu beachten, daß die Ergebnisse je nach voreingestellter VBA – Option COMPARE unterschiedlich sein können. So sind beispielsweise die beiden Zeichenketten des folgenden Vergleichs bis auf die Groß- / Kleinschreibung identisch:

```
MsgBox "Hagen" = "HAGEN"
```

Die Ausgabe der MsgBox meldet keine Übereinstimmung.

Die Option COMPARE kann hier auf zwei unterschiedliche Werte gesetzt werden:

- OPTION COMPARE BINARY unterscheidet zwischen Groß- / Kleinschreibung (obiges Ergebnis FALSCH).
- OPTION COMPARE TEXT bewirkt, daß bei Vergleichen die Groß- / Kleinschreibung ignoriert wird (Das Ergebnis des obigen Beispiels wäre dann WAHR).

 Die Option muß im Modul vor der ersten Prozedur gesetzt werden !

 Der Defaultwert ist OPTION COMPARE BINARY.

```
Option Compare Text
Sub Vergleich ()
Ort1 = "Hagen"
Ort2 = "HAGEN"
 If Ort1 = Ort2 Then
   Gleich = True
 Else
   Gleich = False
 End If
End Sub
```

 Um unnötige falsche Ergebnisse bei derartigen Vergleichen zu vermeiden, sollte man vor dem Vergleich die Zeichenketten „auf das gleiche Level“ bringen – beide in Groß- oder beide in Kleinschrift umwandeln. Siehe dazu die Funktionen UCASE und LCASE weiter im Text.

Teilstrings

Mit Hilfe der Funktionen RIGHT, LEFT und MID lassen sich Teilketten aus Strings herausgelöst werden.

```
Right (Zeichenfolgeausdruck, Länge)
Left (Zeichenfolgeausdruck, Länge)
Mid (Zeichenfolgeausdruck, Startposition, [Länge])
```

RIGHT löst Teilketten aus dem Zeichenfolgeausdruck (Typ STRING) heraus, beginnend mit dem ersten Zeichen rechts, LEFT beginnend mit dem ersten Zeichen links, MID beginnend mit der abgegebenen Position nach rechts um in LÄNGE angegeben Anzahl Zeichen. Wird LÄNGE nicht angegeben, werden alle Zeichen rechts von STARTPOSITION ausgegeben.

```
Set Eingabe = Worksheets("Begriffe").Range("B2")
´enthält die Zeichenkette "Autobahnpolizei"
Ausgabe = Right(Eingabe, 7) 'ergibt:  polizei
Ausgabe = Left(Eingabe, 4) 'ergibt: Auto
Ausgabe = Mid(Eingabe, 5, 4) 'ergibt: bahn
Ausgabe = Mid(Eingabe, 5) 'ergibt: bahnpolizei
```

Leerzeichen in Strings

Die Funktionen RTRIM, LTRIM und TRIM entfernen überflüssige Leerzeichen an unterschiedlichen Positionen eines Strings:

```
RTrim(Zeichenfolgeausdruck)
LTrim(Zeichenfolgeausdruck)
Trim(Zeichenfolgeausdruck)
```

RTRIM entfernt rechts stehende Leerzeichen, LTRIM entfernt führende Leerzeichen, TRIM entfernt sowohl führende als auch alle nachfolgenden Leerzeichen aus Strings.

Insbesondere bei der Bereinigung von Eingabedaten leisten diese Funktionen gute Dienste (siehe Beispiel im nächsten Unterkapitel).

Verknüpfen von Strings

Mit Hilfe des Verkettungsoperators **&** können Variable und Konstanten beliebig verknüpft werden.

```
Ort = "Hagen   "
Vorwahl = "02331"
Nr = "987  01"
Ausgabe = Rtrim(Ort) & ", Tel.: " & Vorwahl & " - " & Trim(Nr)
```

ergibt in der Variablen Ausgabe den Inhalt

Hagen, Tel.: 02331 – 98701

(bitte Leerzeichen bei der Verkettung beachten !)

Groß– / Kleinbuchstaben - Umwandlung

Die Funktionen UCASE und LCASE führen Umwandlungen von Klein- in Großbuchstaben und umgekehrt durch. Eine solche Umwandlung kann insbesondere bei Vergleichen von Zeichenketten (s. oben im Text) unliebsame Ergebnisse zu vermeiden helfen.

Die Funktion Ucase wandelt die Buchstaben einer Zeichenkette in Großbuchstaben um, die Funktion LCase umgekehrt – in Kleinbuchstaben.

Der Vergleich

`"FernUniversität Hagen" = "Fernuniversität Hagen"`

wird unter Standardeinstellungen den Wert FALSE liefern.

Mit Hilfe der Funktion LCASE (oder UCASE) kann die Differenz „glattgebügelt“ werden:

`LCase("FernUniversität Hagen") = Lcase("Fernuniversität Hagen")`

 Die Benutzung dieser Funktionen wird empfohlen, um durch Schreibfehler bewirkte Fehlfunktionen von Kontrollstrukturen (IF – Bedingungen, Schleifen) zu vermeiden.

6 Unterprogrammtechnik

VBA, als eine prozedurale Programmiersprache, zeichnet sich dadurch aus, daß der Programmcode in kleine, unabhängige Einheiten unterteilt ist. Diese Einheiten können sich gegenseitig aufrufen und Parameter übergeben. Ist die Formulierung der Einheiten allgemein genug, können sie immer wieder in unterschiedlichen Programmen benutzt werden.

Es gibt im VBA zwei Syntaxvarianten für solche Einheiten – Prozeduren und Funktionen

6.1 Prozeduren

Wie schon weiter oben im Text beschrieben, beginnt eine Prozedur (auch UNTERPROGRAMM genannt) mit der Anweisung SUB und einem Namen aus max. 255 Zeichen. Dem Namen folgt eine in runde Klammern gesetzte optionale Argumentenliste. Die Prozedur besteht aus zwei Teilen:

- dem Deklarationsteil (auch Prozedurkopf genant)
- dem Programmteil (Programmcode)

Den Abschluß bildet die END SUB – Anweisung:

```
Sub Prozedurname [(Argumentenliste)]

  Anweisungen
  [Exit Sub]
  .
  Anweisungen
End Sub
```

Prozeduren können Aktionen unterschiedlicher Art durchführen, sie können jedoch keine Werte zurückgeben (siehe dazu Funktionen weiter im Text).

Die ARGUMENTENLISTE (optional) besteht aus Deklarationen von Variablen (durch Kommata getrennt). Die Argumente dieser Liste werden FORMALE ARGUMENTE (formale Parameter) genannt.

6.1.1 Aufruf und Parameterübergabe

Sollen beim Aufruf einer Prozedur Parameter übergeben werden, so geschieht die Definition der Argumentenliste in der folgenden Form:

```
[ByVal] Variable1[()] [As Typ] [, [ByVal] Variable2[()] [As Typ]]...
```

Für die Prozedur Testlauf1 werden zwei *formale Parameter* WERT1 und WERT2 definiert:

```
Sub Testlauf1 (Wert1 As String, Wert2 As Long)
   Ort = Wert2 & "  " & Wert1
   MsgBox Ort
End Sub
```

Die Prozedur wird von einer anderen Prozedur aus aufgerufen, wobei beim Aufruf zwei AKTUELLE PARAMETER (auch aktuelle Argumente genannt) angegeben werden müssen. Dies geschieht in der allgemeinen Form:

```
Prozedurname [Argumentenliste]
```

PROZEDURNAME ist der Name der aufzurufenden Prozedur. Benötigt diese keine Argumente, spricht man von einer UNECHTEN PROZEDUR.

Für das obige Beispiel könnte der Aufruf lauten:

```
Sub Aufruf()
Testlauf1 "Hagen", 58084
End Sub
```

Dieser erzeugt als Ergebnis:

Abbildung 21: Ergebnis des Prozeduraufrufs

Sind die formalen und die aktuellen Argumente Variablen und wird in einer Prozedur ein formales Argument verändert, so verändert sich das dazugehörige aktuelle Argument des aufrufenden Programms:

```
Sub Teil1()
Dim Var1 As Integer, Var2 As Integer
Var1 = 5
Var2 = 10
Var3 = Var1 + Var2          (1)

Teil2 Var1, Var2

Var3 = Var1 + Var2          (2)
End Sub

Sub Teil2(Wert1 As Integer, Wert2 As Integer)
Wert1 = 15
Wert2 = 20
End Sub
```

Die Summe bei (1) beträgt 15, die Summe bei (2) ist 35. Obwohl die Variablen Var1 und Var2 in der Prozedur Teil1 nicht geändert wurden, haben sie offensichtlich nach dem Aufruf der Prozedur Teil2 aus Teil1 heraus die Werte der Variablen Wert1 und Wert2 angenommen.

Dieses ist durch die Form der Übergabe der Argumente bedingt. VBA übergibt

- entweder *Call by Reference* – es wird nicht der Wert des aktuellen Arguments übergeben, sondern eine Referenz auf die Speichervariable (Adresse im Arbeitsspeicher)
- oder *Call by Value* – es wird nur der Wert des Arguments übergeben. Änderungen in der aufgerufenen Prozedur haben keine Wirkung, weil dort die Adresse nicht bekannt ist.

Das Problem kann umgangen werden, wenn die Argumentenliste die Wertübergabe erzwungen wird. Dies ist über die Anweisung BYVAL möglich:

```
Sub Teil2(ByVal Wert1 As Integer, ByVal Wert2 As Integer)
Wert1 = 15
Wert2 = 20
End Sub
```

Hier hätte die Wertzuweisung innerhalb von TEIL2 keinen Einfluß auf die Werte von VAR1 und VAR2 in TEIL1.

Die Wertübergabe mittels BYVAL kann in der Argumentenliste für jede Variable einzeln definiert werden.

6.1.2 Optionale Argumente

Es ist nicht immer möglich oder auch erforderlich, alle Argumente an eine Prozedur zu übergeben. Argumente können auch optional definiert werden.

Ein OPTIONALES ARGUMENT wird mit dem Schlüsselwort OPTIONAL gekennzeichnet. Ein so definiertes Argument muß beim Aufruf nicht übergeben werden.

Für die schon weiter oben vorgestellte Prozedur TESTLAUF1 werden zwei *formale Parameter* WERT1 und WERT2 definiert. Der Parameter WERT2 ist optional:

```
Sub Testlauf1 (Wert1 As String, Optional Wert2 As Long)
   Ort = Wert2 & "  " & Wert1
   MsgBox Ort
End Sub
```

Der Aufruf kann dann lauten:

```
Sub Aufruf()
Testlauf1 "Hagen"
End Sub
```

Die Ausgabe:

Das beim Aufruf nicht angegebene, als optional deklarierte Argument liefert in der Ausgabe den Wert 0. Ohne der optionalen Deklaration würde das Programm eine Fehlermeldung ausgeben.

Wird in der Argumentenliste ein Argument als optional deklariert, müssen auch alle nachfolgenden Argumente der Liste optional sein, d.h. optionale Deklarationen sollten am Ende der Liste stehen.

6.2 Benutzerdefinierte Funktionen

Eine benutzerdefinierte Funktion (Funktionsmakro, BDF) sind, wie auch die oben vorgestellten Sub's, Unterprogramme, die mit ihrem Namen aufgerufen werden können. Der Unterschied liegt darin, daß BDF's einen Wert zurückgeben können. Sie lassen sich, wie interne VBA – Funktionen, Variablen oder Konstante in Ausdrücken verwenden.

Die Syntax einer BDF ist der einer Sub ähnlich:

```
[Public][Private][Static]Function Funktionsname [(Argumentenliste)][As Typ]

  Anweisungen
  [Funktionsname = Ausdruck]
  [Exit Function]
  .
  Anweisungen
  [Funktionsname = Ausdruck]
End Function
```

Die ARGUMENTENLISTE ist optional. Werden Argumente definiert, müssen sie beim Aufruf in der richtigen Reihenfolge angegeben werden. Der Aufbau der Argumentenliste entspricht dem einer Sub.

Die Angabe AS TYP bestimmt den Typ der Funktion, genauer gesagt, den Typ des Rückgabewertes. Dieser wird der Funktion innerhalb des Funktionskörpers über die Anweisung

```
Funktionsname = Ausdruck
```

zugewiesen. Funktionstyp und der Typ des Rückgabewertes (in der Zuweisung) müssen übereinstimmen.
Wird der Rückgabewert nicht explizit zugewiesen, gibt eine Funktion einen Standardwert zurück. Dieser entspricht dem definierten Typ der Funktion – numerisch der Wert 0, String ein Leerstring `" "`, Variant dert Typ *Empty*.

Funktionen können

- wie Sub – Prozeduren (mit oder ohne Argumente) aufgerufen werden:

```
Ausgabe
Ausgabe "Hagen", 58084
```

  Bei der Argumentenübergabe sind hier keine Funktionsklammern erforderlich

- in Ausdrücken verwendet werden:

```
MsgBox Ausgabe("Hagen", 58084)
Feld1 = Ausgabe("Hagen", 58084)
Feld1 = Feld2 - Ausgabe("Hagen", 58084)
```

  Bei der Argumentenübergabe sind hier Funktionsklammern erforderlich

 Beispiel für eine benutzerdefinierte Funktion:

```
Function Brutto(ByVal Netto As Single) As Single
   Mwst = 0.16
   If VarType(Netto) = 4 Then
     Ergebnis = Netto + Netto * Mwst
   Else
     MsgBox "Nettoangabe falsch"
   End If
   Brutto = Ergebnis
End Function
```

Nach dem Aufruf:

```
Sub Testen
  MsgBox Brutto(123.75)
End Sub
```

präsentiert die MsgBox das Ergebnis:

7 Ablaufsteuerung I

In diesem Kapitel soll eine kleine Auswahl von einfachen Anweisungen und Operationen vorgestellt werden, die für die Ablaufsteuerung eines VBA – Programms in einer Excel – Arbeitsmappe benutzt werden können.

Mit den hier vorgestellten Verfahren sind schon sehr einfache Programmlösungen möglich.

Weitere Informationen zur Ablaufsteuerung und Navigation in Tabellen und Arbeitsmappen werden in den „Navigationskapiteln“ dieser Unterlage (s. Seiten 68, 73 und **Fehler! Textmarke nicht definiert.**) vorgestellt.

7.1 Cursorposition feststellen

Für die Steuerung des Programms ist es oft wichtig, festzustellen wo sich aktuell der Cursor befindet oder anders ausgedrückt, welche Zelle ist aktuell aktiv. Aus den unterschiedlichen Möglichkeiten ergeben sich unterschiedliche Anweisungen (die ermittelten Positionen werden in den Beispielen jeweils Variablen zugewiesen):

- **Adresse der aktiven Zelle.**

  Die Adresse der aktiven Zelle des aktiven Tabellenblattes wird über die Eigenschaft ACTIVECELL ermittelt:

  ```
  Position = ActiveCell.Address
  ```

  ermittelt.

- **Zeilennummer der aktiven Zelle**

  Soll nur die Zeilennummer der aktiven Zelle ermittelt werden, geschieht es über:

  ```
  Position = ActiveCell.Row
  ```

- **Spaltennummer der aktiven Zelle**

  Fast analog läßt sich die Spaltennummer der aktiven Zelle ermitteln:

  ```
  Position = ActiveCell.Column
  ```

- **Name der Tabelle mit der aktiven Zelle**

  Der Tabellenname wird über die Eigenschaft PARENT der Eigenschaft ACTIVECELL ermittelt:

  ```
  Position = ActiveCell.Parent.Name
  ```

- **Name der Arbeitsmappe**

  Der Name der Arbeitsmappe wird über die Eigenschaft PARENT der Eigenschaft ACTIVESHEET des Windows- oder Application – Objekts ermittel:

  ```
  Position = ActiveSheet.Parent.Name
  ```

7.2 Versetzen des Cursors (Offset – Methode)

Für die Versetzung des Cursors von einer aktiven Zelle (Versetzen der Markierung) in einen anderen Bereich wird die OFFSET – Methode benutzt:

```
Objekt.Offset[(rowOffset, columnOffset)]
```

OBJEKT ist der Bereich, auf den die Methode angewandt werden soll

ROWOFFSET ist die ganzzahlige (positiv, negativ oder 0) Angabe der Zeilenzahl, um die von der aktuellen Position ausgehend verschoben werden soll (ohne Angabe = 0).

COLUMNOFFSET ist die ganzzahlige (positiv, negativ oder 0) Angabe der Spaltenzahl, um die, ausgehend von der aktuellen Position, verschoben werden soll (ohne Angabe = 0).

```
ActiveCell.Offset(1,0).Select
ActiveCell.Offset(0,1).Select
ActiveCell.Offset(2,-3).Select
```

Die erste Anweisung versetzt die Markierung (den Cursor) um eine Zeile nach unten, die zweite Anweisung um eine Spalte nach rechts, die dritte um zwei Zeilen nach unten und drei Spalten nach links.

7.3 Zellen gezielt auswählen

Das gezielte Auswählen von Zellen ist auf unterschiedlichen Wegen möglich.

- Liegt die auszuwählende Zelle / Zellen im aktiven Arbeitsblatt der aktiven Arbeitsmappe, ist die Auswahl einfach:

  ```
  [A1].Select
  ```

  wählt (markiert) die Zelle A1

  ```
  [A1:B5].Select
  ```

  wählt (markiert) den Bereich A1 bis B5.

  Die Auswahl einer einzelnen Zelle ist auch über

  ```
  Cells(Zeile, Spalte).Select
  ```

  beispielsweise mit

  ```
  Cells(3,5).Select
  ```

  für die Zelle E3 möglich.

  Die Auswahl von einzelnen Zellen und Zellbereichen ist ebenfalls über die Range – Eigenschaft des Tabellenobjekts möglich (absolute Positionierung):

  ```
  Range(Adr1 [[, Adr2] ...])
  ```

  beispielsweise

  ```
  Range("A2")
  Range("A1:C7")
  Range("B2, C3:D4, E7, E9.F10")
  ```

- Liegt der auszuwählende Bereich in einem nicht aktiven Tabellenblatt der aktiven Arbeitsmappe, muß zuerst das gewünschte Tabellenblatt aktiviert werden:

  ```
  Worksheets(Tabellenname).Select
  ```

  und anschließend die Auswahl getroffen werden:

  ```
  Worksheets("Tabelle2").Select
  Cells(5,3).Select
  ```

  aktiviert Tabelle2 und markiert darin die Zelle C5.

- Liegt der auszuwählende Bereich in einer nicht aktiven Arbeitsmappe, wird zuerst über

  ```
  Workbooks(Arbeitsmappenname).Activate
  ```

  die Arbeitsmappe aktiviert, anschließend darin ein Tabellenblatt und danach der Zellbereich:

  ```
  Workbooks("DATEN2.XLS").Activate
  Worksheets("Sommer99").Select
  Cells(5,3).Select
  ```

  oder

  ```
  Workbooks("DATEN3.XLS").Activate
  Worksheets("Winter99").Select
  Range("A5:C7").Select
  ```

7.4 Inhalte in einzelne Zellen eintragen

Die Wertzuweisung an einzelne Zellen erfolgt mit Hilfe der VALUE – Eigenschaft. Die folgende Beispiele zeigen die einfachste Form einer solchen Zuweisung:

```
ActiveCell.Value = 333
ActiveCell.Value = "Hagen"
ActiveCell.Value = 25.75
Range("A5").Value = 55
```

Ist die Zelle nicht aktiv, muß sie vor der Wertzuweisung aktiviert werden:

```
Range("A5").Activate
ActiveCell.Value = 333
```

Wird die Eigenschaft VALUE nicht angegeben, so wird systemseitig automatisch VALUE angenommmen:

```
ActiveCell = 333
ActiveCell = "Hagen"
ActiveCell = 25.75
Range("A5") = 55
```

7.5 Formeln in Zellen eintragen

Der Eintrag von Formeln in Zellen ist über die FORMULA – Eigenschaft möglich.

```
ActiveCell.Formula = "=A1 + C3"
Cells(2, 2).Formula = "=A1 + C3"
Sheets("Tabelle1").Cells(2, 3).Formula = "=A1 + C3"
```

7.6 Ausschneiden

Das Ausschneiden von Zellinhalten (Übertragen des Inhalts in die Zwischenablage) ist mittels der CUT – Methode möglich:

```
Object.Cut(destination)
```

Der Zellinhalt bleibt dabei im Tabellenblatt sichtbar, lediglich ein Laufrahmen um die Zelle signalisiert den Vorgang.

```
ActiveSheet.Cells(3, 3).Cut
ActiveSheet.Range("B2").Cut
ActiveSheet.Range("B2:C3").Cut
```

Soll der ausgeschnittene Inhalt im gleichen Schritt an einer anderen Stelle eingefügt werden, wird die DESTINATION – Angabe benutzt:

```
ActiveSheet.Cells(3, 3).Cut Destination:=Range("C4")
ActiveSheet.Range("B2").Cut Destination:=Cells(2, 3)
ActiveSheet.Range("B5:C6").Cut Destination:=Range("D1")
```

7.7 Kopieren

Der Unterschied zum Ausschneiden besteht beim Kopieren darin, das der Inhalt der kopierten Zellen erhalten bleibt. Die Syntax ist, bis auf die Methode (COPY) gleich:

```
ActiveSheet.Cells(3, 3).Copy
ActiveSheet.Range("B2").Copy
ActiveSheet.Range("B2:C3").Copy
ActiveSheet.Cells(3, 3).Copy Destination:=Range("C4")
ActiveSheet.Range("B2").Copy Destination:=Cells(2, 3)
ActiveSheet.Range("B5:C6").Copy Destination:=Range("D1")
```

✘ Die Inhalte der Zielzellen werden, wie bei manuellen Kopiervorgängen, ohne Vorwarnung überschrieben.

✘ Im Unterschied zum manuellen Kopieren ist der Inhalt der Zwischenablage nach dem Einfügen leer. Mehrmalige Kopiervorgänge müssen also programmtechnisch geregelt werden.

7.8 Einfügen

Neben der schon oben beschriebenen Möglichkeit, Inhalte der Zwischenablage in Zellen / Zellbereiche einzufügen, ist es zusätzlich möglich mittels der PASTE – Methode Zellen mit dem Inhalt der Zwischenablage zu belegen. Die Zwischenablage sollte vorher mir CUT oder COPY einen Inhalt erhalten:

```
ActiveSheet.Cells(1, 1).Cut
ActiveSheet.Paste Destination:=Cells(1, 2)
ActiveSheet.Cells(2, 1).Copy
ActiveSheet.Paste Destination:=Range("B3")
ActiveSheet.Cells(3, 1).Cut
ActiveSheet.Paste Destination:=Worksheets("Tabelle2").Range("A4")
```

✘ Wenn die Zwischenablage leer ist, fügt PASTE leere Zellen ein. Auf diese Weise kann in Tabellen leerer Platz geschaffen werden.

7.9 ASCII – Werte / ASCII – Zeichen

Jedem Zeichen des ASCII – Codes ist ein numerischer Wert zugeordnet.

Dieser Wert läßt sich über die Funktion ASC ermitteln:

```
Ausgabe = Asc("A")
```

Der umgekehrte Weg – die Ermittlung des zu einem numerischen Wert gehörenden ASCII – Zeichens ist über die Funktion CHR möglich:

```
ActiveCell.Value = Chr(75)
```

belegt die aktive Zelle mit dem Buchstaben **K**.

Beide Funktionen lassen sich kombinieren und in Funktionen und Ausdrücken benutzen:

```
ActiveCell.Value = Chr(Asc("D") + 4)
ActiveCell.Value = Chr(Asc("D") - 2)
```

Das Ergebnis der ersten Anweisung ist ein **H** in der aktiven Zelle, die zweite liefert ein **B** in der aktiven Zelle.

 Beide Anweisungen werden gerne zum einfachen Verschlüsseln von Passwörtern benutzt. Das folgende Beispiel demonstriert eine sehr einfache Verschlüsselung unter Anwendung einiger bisher schon vorgestellter Elemente:

```
Sub testen()
Eingabe = "FernUni"
Weite = Len(Eingabe)
Ausgabe = ""
  For i = 1 To Weite
    Ausgabe = Ausgabe + Chr(Asc(Mid(Eingabe, i, 1)) + 4)
  Next
 MsgBox Eingabe & Chr(13) & Chr(13) & Ausgabe, , "Ergebnis"
End Sub
```

Das Ergebnis, über MsgBox ausgegeben:

Durch den umgekehrten Weg kann eine solche Verschlüsselung wieder rückgängig gemacht werden

8 Dialoge (Teil I)

Für die Programmsteuerung ist es oft nötig, Meldungen an Benutzer auszugeben bzw. Eingaben des Benutzers ins Programm zu übernehmen.

Diese Aufgaben können vom System zur Verfügung gestellte Dialogfunktionen oder individuell durch den Benutzer definierte Oberflächen übernehmen.

Die Handhabung der System - Elemente wird im folgenden näher beschrieben. Benutzerdefinierte Dialoge werden in einem späteren Kapitel vorgestellt.

8.1 MsgBox

Die in den bisherigen Beispielen schon öfter benutzte MSGBOX ist eine den Programmablauf unterbrechende Funktion, die eine Meldung in einem Dialogfeld anzeigt und auf die Auswahl einer Schaltfläche wartet.

Die Funktion hat die Syntax:

```
MsgBox (Prompt [, Buttons][,Title][,Helpfile][,Context]
```

PROMPT	ist ein Zeichenfolgeausdruck mit der maximalen Länge von 1024 Zeichen, der als Meldung im Dialogfeld der MSGBOX erscheint. Soll der Meldungstext aus mehreren Zeilen bestehen, müssen die Zeilen durch manuelles Einfügen von Zeilenumbrüchen mittels der Funktion CHR (Zeichen CHR(13)) umbrochen werden
BUTTONS	ist ein numerischer Ausdruck mir einem kombinierten Wert, der die Anzahl und den Typ der Schaltflächen, ein evtl. verwendetes Symbol, die aktivierte Schaltfläche und die Bindung des Dialogfeldes.
TITEL	ist ein Zeichenfolgeausdruck, der als Titel der Dialogbox erscheinen soll.
HELPFILE	Nur in Verbindung mit CONTEXT zu verwenden – definiert die kontextbezogene Hilfedatei für das Dialogfeld-
CONTEXT	Numerischer Ausdruck, der dem Hilfethema in der unter HELPFILE zugeordneten Hilfedatei zugeordnet ist.

Die einfachste Form der Anwendung ist die Angabe der Funktion nur mit PROMPT:

```
MsgBox "Falsche Eingabe"
MsgBox Netto * 1,16
MsgBox ActiveCell.Value + 5
```

Wird das Argument BUTTONS verwendet, müssen entweder numerische Werte oder Konstanten zur Definition der Schaltflächen angegeben werden:

Schaltflächen:

Wert	Konstante	Funktion
0	vbOKOnly	(Voreinstellung) Schaltfläche OK erzeugen
1	vbOKCancel	OK und ABBRECHEN erzeugen
2	vbAbortRetryIgnore	ABBRECHEN, WIEDERHOLEN IGNORIEREN erzeugen
3	vbYesNoCancel	JA, NEIN, ABBRECHEN erzeugen
4	VbYesNo	JA, NEIN erzeugen
5	VbRetryCancel	WIEDERHOLEN, ABBRECHEN erzeugen

Dialogfeldsymbole:

Wert	Konstante	Funktion
16	vbCritical	Stop – Symbol
32	vbQuestion	Fragezeichen - Symbol
48	vbExclamation	Ausrufezeichen – Symbol
64	vbInformation	Info - Symbol

Aktivierte Schaltflächen:

Wert	Konstante	aktivierte Schaltfläche
0	vbDefaultButton1	erste Schaltfläche
256	vbDefaultButton2	zweite Schaltfläche
512	vbDefaultButton3	dritte Schaltfläche

Bindung (Modalverhalten) des Dialogfeldes:

Wert	Konstante	Funktion
0	vbApplicationModal	Gebunden an die Anwendung. Die aktuelle Anwendung kann nur fortgesetzt werden, wenn der MSGBOX – Dialog beendet wird. Alle anderen Anwendungen sind nicht betroffen.
256	vbSystemModal	Systemgebunden – alle Anwendungen werden angehalten, bis der MSGBOX – Dialog beendet ist.

Bei der Definition der Schaltflächen und Symbole können entweder die Wert oder die Konstanten benutzt werden.

Soll beispielsweise ein Dialogfeld mit einer JA-, einer NEIN – Schaltfläche, versehen mit dem Fragezeichen – Symbol und der aktivierten NEIN – Schaltfläche erzeugt werden, geschieht es in der folgenden Form:

```
vbYesNo + vbQuestion + vbDefaultButton2
```

oder

```
4 + 32 + 256
```

oder einfach

```
292
```

wobei die letzte Version die Summe der Einzelwerte ist. Hier ist allerdings kaum erkennbar, was genau definiert wurde.

Ein Dialogfeld mit einer JA-, einer NEIN – Schaltfläche, versehen mit dem Fragezeichen – Symbol und der aktivierten NEIN – Schaltfläche, welches die Meldung „*Soll der Lauf abgebrochen werden ?“* anzeigt und den Titel „*Nachfrage*“ besitzt, kann durch die folgende Anweisung erzeugt werden:

```
MsgBox "Soll der Lauf abgebrochen werden", vbYesNo + vbQuestion + _
vbDefaultButton2, "Nachfrage"
```

Soll der durch das Dialogfeld gelieferte Rückgabewert ausgewertet werden, muß die Funktionsschreibweise (mit Argumentenklammern !) benutzt werden:

```
Aus = MsgBox("Soll der Lauf abgebrochen werden", vbYesNo + vbQuestion + _
vbDefaultButton2, "Nachfrage")
```

Der der Variablen AUS zugewiesene Rückgabewert kann anschließend ausgewertet werden.

Für die Rückgabe des Dialogfeldes gelten die folgenden Werte oder Konstanten:

Schaltfläche	Wert	Konstante
OK	1	vbOK
ABBRECHEN	2	vbCancel
ABBRUCH	3	vbAbort
WIEDERHOLEN	4	vbRetry
IGNORIEREN	5	vbIgnore
JA	6	vbYes
NEIN	7	vbNo

Die Auswertung des Rückgabewertes wird meistens in einer Abfrage realisiert:

```
If Aus = vbYes then....
```

oder

```
If Aus = 6 then....
```

Der Umweg über die Variable kann gespart werden, wenn in die Abfrage die MSGBOX – Funktion aufgenommen wird:

```
Sub Abbruch()
If MsgBox("Soll der Lauf abgebrochen werden", vbYesNo + vbQuestion + _
vbDefaultButton2, "Nachfrage") = vbYes Then
   'Funktionsaufruf der Abbruchfunktion
   Abbruch
End If
End Sub
```

Das Beispiel erzeugt das folgende Dialogfeld

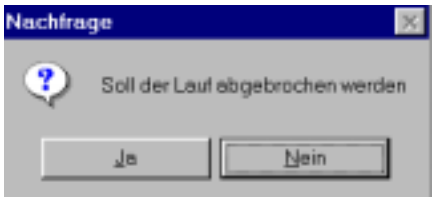
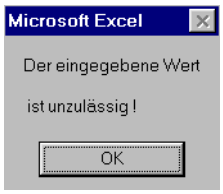


Abbildung 22: MsgBox - Dialogfeld

 Zeilenumbruch im Text der MSGBOX mit Hilfe der Funktion CHR:

```
MsgBox "Der eingegebene Wert" + Chr(13) + "ist unzulässig ! "
MsgBox "Der eingegebene Wert" & Chr(13) & "ist unzulässig ! "
MsgBox "Der eingegebene Wert" & Chr(13) & Chr(13) & "ist unzulässig ! "
```

Die letzte Anweisung erzeugt die Ausgabe:



8.2 InputBox

Benutzereingaben können über eine INPUTBOX getätigt werden. INPUTBOX wird von VBA als Funktion und als Methode zur Verfügung gestellt.

8.2.1 Die Funktion InputBox

Die Funktion INPUTBOX hat eine Syntax, die ähnlich ist der Syntax der im vorherigen Kapitel vorgestellten Funktion MSGBOX:

```
InputBox(Prompt [,Titel][,default][,xpos][,ypos]
[,helpfile][,context])
```

Die Argumente entsprechen denen der MSGBOX – Funktion.

Hinzukommen die Argumente

DEFAULT	ist ein Zeichenfolgeausdruck, der als Vorbelegung für das Eingabefeld gilt, wenn keine Eingabe erfolgt. Wird dieses Argument nicht angegeben, ist das Eingabefeld leer.
XPOS	ist ein numerischer Ausdruck für die x - Position des Dialogfeldes im Verhältnis zur oberen linken Bildschirmecke. Wir dieses Argument nicht angegeben, so wird das Dialogfeld horizontal zentriert.
YPOS	ist ein numerischer Ausdruck für die y - Position des Dialogfeldes im Verhältnis zur oberen linken Bildschirmecke. Wir dieses Argument nicht angegeben, so wird das Dialogfeld etwa 1/3 vom oberen Bildrand positioniert.

➡ Die Funktion gibt immer eine Zeichenfolge zurück. Wird die Eingabe unterbrochen, ist die zurückgegebene Zeichenfolge leer.

✘ Wurde DEFAULT definiert und die Eingabe unterbrochen, so ist die zurückgegebene Zeichenfolge ebenfalls leer.

➡.Ausgelassene Argumente sind durch Kommata als Platzhalter zu ersetzen.

```
Eingabe = InputBox(,,,830,950) _
```

Hier wurden die Argumente PROMPT, TITLE und DEFAULT nicht definiert (ausgelassen). Werden Argumente am Listenende ausgelassen, gilt diese Regel nicht.

💾 Die Anweisung

```
Eingabe = InputBox("Bitte Anzahl Studenten angeben", _
     "Studentenstatistik", 0, 830, 950)
```

erzeugt das Dialogfeld:

Abbildung 23: Dialogfeld InputBox

✘ Der eingegebene Wert wird mit Ok übernommen.

✘ Die Werte der Argumente XPOS und YPOS (siehe Beispiel) werden in Points angegeben. Ein Point = 1/72 Zoll (0,35 mm).

8.2.2 Die Methode InputBox

Die Methode INPUTBOX unterscheidet sich von der Funktion INPUTBOX vor allem dadurch, daß sie eine Möglichkeit bittet, den Typ des eingegebenen Wertes zu definieren. Hierzu wird die Syntax um das Argument TYPE erweitert:

```
Object.InputBox(Prompt [,Titel][,default][,Left][,Top]
[,helpfile][,context][,Type])
```

Wird dieses Argument nicht angegeben, so wird der Eingabewert als Zeichenfolge interpretiert und bei der Übergabe an eine Zelle je nach Form als Zahl, Text, boolscher Wert oder Formel interpretiert.

Dem Argument TYPE können folgende Werte zugewiesen werden:

Wert	Typ
0	Formel
1	Numerisch
2	Text
4	Logisch
8	Zellbezug (Bereichs – Objekt)
16	Fehlerwert (z.B. #NV)
64	Wertematrix

Der definierte Wert des Arguments kann sich auch aus einer Summe von Werten ergeben (siehe z.B. MSGBOX).

Die Argumente LEFT und TOP definieren im Unterschied zu XPOS und YPOS der Funktion jeweils den Abstand des Dialogfeldes in Points zum linken bzw. rechten Rand des Bildschirms

```
ActiveCell.Value = Application.InputBox("Bitte Anzahl Studenten angeben", _
     "Studentenstatistik", 0, , , , 1)
```

 Ohne Angabe von OBJEKT (hier *Application*) geht VBA davon aus, daß die Funktion INPUTBOX benutzt wird und gibt eine Fehlermeldung wegen falscher Anzahl Argumente aus !

 Wird für das Argument TYP der Wert 8 gesetzt, erlaubt die als Methode aktivierte INPUTBOX die Auswahl einer Zelle oder eines Zellbereichs per Maus anstelle einer manuellen Eingabe in das Eingabefeld. Die Methode gibt eine Range – Objekt zurück, welches mit Werten belegt werden kann.

 Das folgende Beispiel demonstriert die Anwendung der InputBox – Methode zum Füllen markierter Zellbereiche mit Werten. Über das Dialogfeld wird der Benutzer aufgefordert, einen Zellbereich zu markieren. Dieser wird über die an das von INPUTBOX zurückgegebene RANGE – Objekt angehängte VALUE – Eigenschaft, mit einem Wert belegt.

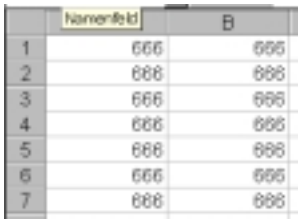
Mit einem ähnlichen Verfahren können Zellbereiche mit identischen oder (soweit beispielsweise in einer Schleife erzeugt) unterschiedlichen Werten gefüllt werden. Insbesondere sind als einzutragende Werte auch Formeln möglich.

```
Sub Eintrag ()
Anzahl = 666
Application.InputBox("Bitte Zielbereich markieren",,,,,,,8).Value = Anzahl
End Sub
```

Die Anweisungen des Beispiels erzeugen das Dialogfeld (hier schon nach der Markierung des Zellbereichs:

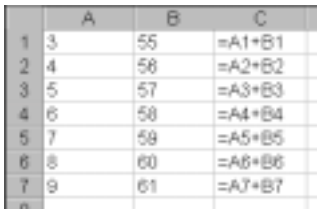
Abbildung 24: INPUTBOX mit dem Ergebnis einer Bereichsmarkierung

Nach Bestätigung mit Ok wird den Zellen des markierten Bereichs der gewünschte Wert zugewiesen:



Wird nach dem gleichen Verfahren versucht, eine Formel in einen markierten Bereich einzutragen, und enthält diese Formel relative Zelladressen, so werden diese angepaßt. Im folgenden Beispiel wurde die Formel in den markierten Bereich C1:C7 automatisch eingetragen. Ergebnis – siehe unten:

```
Sub Eintrag()
Anzahl = "=A1+B1"
Application.InputBox("Bitte Zielbereich markieren",,,,,,, 8).Value = Anzahl
End Sub
```



Soll die Relativierung der Adressen vermieden werden, müssen an entsprechenden Stellen der Formeln absolute Zellbezüge (Adressen) verwendet werden.

9 Tabellennavigation

Die Navigation in Tabellenblättern, konkreter - die Cursor – Bewegung innerhalb einer Tabelle besteht nicht nur aus dem Versetzen des Cursors in eine bestimmte Zelle oder einen Zellbereich, sondern erfordert oft Bewegungen des Cursors relativ zu einer ermittelten Position. Hierzu sind öfters kompliziertere Verfahren nötig, als die im Kapitel 7 schon vorgestellten.

Zum Bereich Tabellennavigation gehören Methoden und Eigenschaften, deren wichtigste aus der folgenden alphabetisch nach Namen sortierten Übersicht ersichtlich sind:

Name	Typ	Funktion
Activate	Methode	Aktiviert eine Zelle, setzt den Cursor in eine Zelle.
Column	Eigenschaft	Ermittelt die Nummer der ersten Spalte eines Range – Objekts (gibt numerischen Indexwert aus)
Columns	Methode	Gibt ein Range – Objekt zurück (einzelne Spalte oder Spaltenauflistung)

Name	Typ	Funktion
COUNT	Eigenschaft	Gibt die Anzahl von Elementen zurück, z.B. Anzahl Zellen eines markierten Bereichs.
ENTIRECOLUMN	Eigenschaft	Gibt ein Range – Objekt für eine oder mehrere Spalten aus.
ENTIREROW	Eigenschaft	Wie oben, jedoch für Zeilen.
NEXT	Eigenschaft	Gibt ein Range – Objekt zurück – die Zelle rechts von der aktuellen.
OFFSET	Methode	Liefert (Range – Objekt) einen versetzten Bereich zum angegebenen.
PREVIOUS	Eigenschaft	Gibt ein Range – Objekt zurück – die Zelle links von der aktuellen.
ROW	Eigenschaft	Liefert die Nummer der ersten Zeile eines Bereichs (Range – Objekts).
ROWS	Methode	Wie Columns, hier bezogen auf Zeilen
SELECT	Methode	Wählt (markiert) einen Zellbereich aus.

Da diese Eigenschaften und Methoden auf Range – Objekten arbeiten, müssen diese erst ermittelt werden. Dabei ist zu beachten, daß VBA kein Objekt für eine einzelne Zelle kennt. Einzelne Zellen zählen als Sonderfall eines Bereichs (Range).

Für die Ermittlung der Range – Objekte werden oft zwei den Objekten APPLICATION und WORKSHEET zugeordneten Zugriffsmethoden verwendet: RANGE und CELLS. Die Anwendung ist schon im Kap 7.3 erläutert worden.

9.1 Absolute Positionierung auf Zellen und Zellbereiche

Für die absolute Positionierung wird entweder die Range – Methode oder die Cells – Eigenschaft verwendet. In beiden Fällen besteht die Möglichkeit, sowohl auf einzelne Zellen oder Zellbereiche zu positionieren.

9.1.1 Positionierung über die Range – Methode

Die Range Methode hat bei der Positionierung zwei Versionen:

```
Objekt.Range(Cell)
Objekt.Range(Cell1, Cell2)
```

Das CELL – Argument kann eine einzelne Zelladresse, in der Excel – Standardform (A1, B3, K55 usw.), eine Bereichsadresse (A1:B5, G45:K55) oder eine durch Kommata getrennte Liste von Adressen. Die Angaben werden in Hochkommata eingeschlossen.

```
Worksheets("Testtabelle").Range("M4")
Worksheets("Testtabelle").Range("M4:N7")
Worksheets("Testtabelle").Range("B1:D3 D1:E3")
Worksheets("Testtabelle").Range("D4, B3, A2, E2:F3")
```

 Die erste Anweisung arbeitet mit einer einzelnen Zelle, die zweite mit dem angegebenen Zellbereich. Die dritte Anweisung verwendet den *Schnittmengen – Operator* (Leerzeichen) und arbeitet mit der Schnittmenge der angegebenen Bereiche (hier D1:D3). Die vierte Anweisung arbeitet mit allen angegebenen Bereichen (wie Mehrfachmarkierung).

Die zweite Syntaxform besitzt zwei Argumente, die Zellen in der oberen linken und unteren rechten Ecke des zu definierenden Bereichs stehen. Cell1 und Cell2 können Adressen von

Einzelzellen oder Zellbereichen sein. Überschneiden sich die Bereiche, wird die Vereinigungsmenge genommen.

```
Worksheets("Testtabelle").Range("B1", "D3")
Worksheets("Testtabelle").Range("B1:C3", "C1:E5")
```

 Die erste Anweisung arbeitet mit dem Zellbereich B1:D3, die zweite mit dem Bereich, der aus der Vereinigung der Bereiche B1:C3 und C1:E5, also dem Bereich B1:E5.

 Solange im aktiven Tabellenblatt gearbeitet wird, benötigt die Range – Methode das Worksheet – Objekt nicht. Damit Verkürzen sich die Anweisungen, z.B. zu:

```
Range("M4:N7")
Range("B1:D3 D1:E3")
Range("B1:C3", "C1:E5")
```

9.1.2 Positionierung über die Cells - Eigenschaft

Im Unterschied zu der Positionierung über die RANGE – Eigenschaft bietet diese Alternative die Möglichkeit, bei der Positionierung Zeilen- und Spaltenindizes zu verwenden, was insbesondere für berechnete Zelladressen oder die Benutzung von Variablen für Zelladressen ein großer Vorteil ist. Auch hier sind mehrere Syntaxvarianten möglich:

```
Objekt.Calls(RowIndex, ColumnIndex)
Object.Cells(Index)
Object.Cells
```

- **Manuelle Eingabe der Zellindizes**

  Die erste Variante wird zur Ermittlung von Adressen einzelner Zellen benutzt. ROWINDEX steht für den Zeilenindex, COLUMNINDEX für den Spaltenindex der Zelle. Für die Zuweisung eines Wertes an die Zelle B2 der Tabelle1 steht dann beispielsweise

  ```
  Worksheets(1).Cells(2,2).Value = 233
  ```

  Die zweite Variante ermittelt ebenfalls eine Zelle. INDEX ist hier die „laufende Nummer“ einer Zelle im Tabellenblatt. Die Zählung beginnt mit der Zelle A1 (Index = 1), für letzte Zelle der ersten Zeile gilt Index = 256, A2 hat den Indexwert 257, usw.. Somit hätte die obige Anweisung in dieser Syntaxvariante die Form:

  ```
  Worksheets(1).Cells(258).Value = 233
  ```

  Die dritte Variante gibt eine Liste aller Zellen eines Tabellenblattes zurück. Sie wird benutzt für Methoden und Eigenschaften die Formatierungen an Zellen übergeben, aber nicht mit WORKSHHEET – Objekten (Tabellenblätter), sondern nur mit RANGE – Objekten arbeiten.

  Für das Entfernen der Hintegrundmuster aus Zellen eines Tabellenblattes könnte (!) die Anweisung

  ```
  Worksheets(1).Interior.Pattern = xlNone
  ```

  benutzt werden, die jedoch nicht den gewünschten Effekt liefert (Laufzeitfehler 438). Weil die INTERIOR – Eigenschaft nur RANGE – Objekte akzeptiert muß mit CELLS ein Range – Objekt erzeugt werden:

  ```
  Worksheets(1).Cells.Interior.Pattern = xlNone
  ```

   Solange im aktiven Tabellenblatt gearbeitet wird, benötigt die CELLS – Eigenschaft, wie die schon oben vorgestellte RANGE - Methode das WORKSHEET – Objekt nicht.

- **Berechnete Zellindizes**

  Die numerische Form der Indizierung in der CELLS – Eigenschaft macht, wie schon oben erwähnt, Berechnungen von Indizes möglich:

```
Sub Testen()
Dim i As Integer
  For i = 1 To 5
    Worksheets(1).Cells(i + 2, i).Value = 233
  Next
End Sub
```

 Ist das Ergebnis bei der Berechnung eine Dezimalzahl, werden die Werte vor der Zuweisung ab- oder aufgerundet.

- **Kombinationen von Cells und Range**

  Durch die Kombination von CELLS und RANGE ist es möglich, Zellbereiche mittels berechneter Indizes zu adressieren:

```
Sub Testen()
Dim i As Integer
  For i = 5 To 7
   Worksheets(1).Range(Cells(2, 2), Cells(i + 2, i)).Interior.Pattern = 2
  Next
End Sub
```

  Die Anweisungen dieses Beispiels belegen den Zellbereich B2:G9 mit einem Muster. Der Startwert ist fest vorgegeben (*Cells(2,2) = Zelle B2*), das zweite Argument der RANGE – Methode wird berechnet.

9.2 Relative Positionierung auf Zellen und Zellbereiche

Für die relative Positionierung innerhalb eines Tabellenblattes muß zuerst eine Ausgangsposition (Startposition) festgelegt werden. Ausgangsposition ist in den meisten Fällen ein aktives Objekt.

Dazu muß ermittelt werden, welches Objekt aktuell aktiv ist. Für die relative Positionierung auf Zellen oder Zellbereiche handelt es sich i.d.R. um die Objekte *Zelle, Tabellenblatt*, *Fenster und Arbeitsmappe*, die über die folgenden Zugriffseigenschaften ermittelt werden können:

ACTIVECELL	Aktive Zelle (Range – Objekt)
ACTIVESHEET	Aktives Tabellen- oder Diagrammblatt (Worksheet – Objekt)
ACTIVEWINDOW	Aktives Fenster (Window – Objekt)
ACTIVEWORKBOOK	Aktive Arbeitsmappe (Workbook – Objekt)

Insbesondere bei Aktionen, die von aktiven Zellen ausgehen, müssen oft zusätzlich noch die Zeilen- und Spaltenindizes ermittelt werden. Dieses ist über die Row- und Column – Eigenschaften des Zellobjekts möglich:

```
ActiveCell.Row
```

ermittelt den Zeilenindex einer aktiver Zelle

```
ActiveCell.Column
```

ermittelt den Spaltenindex einer aktiver Zelle

✘ Wird diese Methode auf einen Zellbereich angewandt, können die Indizes der obersten Zeile und der äußerst linken Spalte des Zellbereichs ermittelt werden:

```
Range("B3:C5").Row      ergibt 3
Range("B3:C5").Column   ergibt 2
```

Mit so ermittelten Positionen kann relativ positioniert werden:

```
Cells (ActiveCell.Row + 7, ActiveCell.Column - 1).Value = "Gesamt: "
```

Auch die im Kap. 7.2 vorgestellte OFFSET – Methode kann bei der relativen Positionierung geeignet. In der Sub:

```
Sub Testen()
Dim i As Integer
  For i = 1 To 3
    ActiveCell.Offset(i + 2, ActiveCell.Column).Interior.Pattern = 2
  Next
End Sub
```

werden beispielsweise bei aktiver Zelle C3 die Zellen F6:F8 mit einem Muster belegt. (Näheres zu OFFSET – siehe Kap. 7.2).

9.3 Zugriff auf Tabellenblätter und Arbeitsmappen

Sobald Operationen auf Zellen oder Zellbereichen nicht im aktiven Tabellenblatt oder eienr aktiven Arbeitsmappe ablaufen sollen oder das aktive Tabellenblatt oder Arbeitsmappe nicht genau bekannt sind, wird es nötig sein, eines der Elemente selbst zu bestimmen.

- **Zugriff auf Tabellenblätter**

  Für die Auswahl eines Tabellenblattes gibt es im VBA zwei Alternativen:

  1. Auswahl über den *Index* des Worksheets – Objekts Tabellenblatt
  2. Auswahl über den *Namen* des Worksheets – Objekts

  Alternative 1

  Der Index eines Tabellenblattes ist die relative Position des Tabellenblattes in der Worksheets – Auflistung.

  ```
  Worksheets(3).Cells(3,5).Value = 5
  ```

  setzt den Wert 5 in die Zelle E3 des dritten Tabellenblattes der aktiven Arbeitsmappe ein.

  ✘ Beim Zugriff über den Indexwert ist Vorsicht geboten: wird im Programm ein Tabellenblatt gelöscht, eingefügt oder verschoben, verändern sich die Indexwerte der Tabellenblätter in der Worksheets – Auflistung.

  ✘ Die Indexwerte der Tabellenblätter können in Schleifenkonstrukten benutzt werden, wenn mehrere Tabellenblätter einer Arbeitsmappe gleichen Operationen unterzogen werden sollen:

  ```
  Sub Testen()
  For n = 1 To Worksheets.Count
    Worksheets(n).Cells(2, 2).Formula = "=A1*0.5"
  Next
  End Sub
  ```

oder einfacher (ohne Zählvariable):

```
Sub Testen()
For Each TabBlatt In Worksheets
  TabBlatt.Cells(2, 2).Formula = "=A1*0.5"
Next
End Sub
```

Beide Subs setzen die Formel =A1*0.5 in die Zelle B2 aller Tabellenblätter der aktiven Arbeitsmappe ein.

Alternative 2

Die Blattauswahl über Namen des Tabellenblattes ist auf drei Wegen möglich:

1. Das schon weiter oben beschriebene Aktivieren des Tabellenblattes:
   ```
   Worksheets("Tabelle1").Activate
   Worksheets("WS99/2000").Activate
   ```
2. Über direkte Angabe der Objektreferenz, die allerdings relativ lange Anweisungen erzeugen kann:
   ```
   Worksheets("Tabelle1").Cells(3,5).Value = 233
   ```
3. Über Objektvariablen (vorher deklarieren !)
   ```
   Sub Testen()
   Dim TBName
   Set TBNAme = Worksheets("Statistik 99")
   TBName.Cells(3,5).Value = 233
   End Sub
   ```

✘ Alle drei Lösungen sind gleichwertig, wegen der kürzeren Schreibweise wird allerdings oft die Auswahl über Objektvariable (3) bevorzugt.

- **Zugriff auf Arbeitsmappen**

  Sind mehrere Arbeitsmappen geöffnet, wird i.d.R. nur in einer gearbeitet (eine Arbeitsmappe ist aktiv). Allerdings muß ein Wechsel zwischen Arbeitsmappen möglich sein. Über die Auflistung WORKBOOKS kann der Wechsel realisiert werden:

  ```
  Workbooks("Mappe1.xls").Worksheets("Tabelle2").Activate
  Workbooks("Probe.xls").Worksheets("Tabelle1").Activate
  ```

  Weil WORKBOOKS eine Auflistung ist, kann auch mit Indizes gearbeitet werden:

  ```
  Workbooks(1).Worksheets("Tabelle2").Activate
  Workbooks(2).Worksheets("Tabelle1").Activate
  ```

  Dabei stehen die Indizes für die Reihenfolge, in der die Arbeitsmappen geöffnet wurden.

10 Manipulation von Zellen und Zellbereichen

Der Zugriff auf Zellen oder Zellbereiche ist nicht trivial, weil VBA mit zahlreichen, inhaltlich ähnlichen Begriffen und Objekten arbeitet. Viele Operationen sind auf unterschiedlichen Wegen realisierbar.

Einige der im folgenden vorzustellenden Techniken sind schon in früheren Kapiteln, zumindest in den Beispielen in kurzer Form vorgestellt werden. Sie werden hier, der Vollständigkeit halber evtl. nochmals erwähnt.

10.1 Auswahl von Zellen und Zellbereichen

Für die Auswahl (markieren, selektieren) von einzelnen Zellen oder Zellbereichen wird die in vielen Beispielen schon verwendetet Methode ACTIVATE verwendet.

```
Range("A3").Activate
Range("B1:D4").Activate
```

Die gleiche Aufgabe erfüllt die Methode SELECT:

```
Range("A3").Select
Range("B1:D4").Select
```

 Zu beachten sind hier die im Kap. 9.1.1 beschriebenen Alternativen der Definition von Argumenten zu RANGE.

 Auch in markierten (ausgewählten) Zellbereichen ist jeweils nur eine Zelle aktiv. Die Adresse dieser Zelle kann für Wert- oder Formatzuweisungen mit

```
ActiveCell.Address
```

ermittelt werden.

Die Adresse des gesamten markierten Zellbereichs ermittelt man mit

```
ActiveWindow.RangeSelection.Addresss
```

Soll die ganze Zeile oder Spalte, in der eine aktive Zelle steht ausgewählt (markiert) werden, werden die Zugriffseigenschaften ENTIREROW und ENTIRECOLUMN benutzt:

```
Worksheets("Tabelle1").Range("B3").EntireRow.Select
Worksheets("Tabelle1").Range("B3").EntireColumn.Select
Worksheets("Tabelle1").Range(ActiveCell.Address).EntireColumn.Select
```

Die erste Anweisung markiert die Zeile 3, die zweite markiert die Spalte B, die dritte die Spalte mit der aktiven Zelle (alle im Tabellenblatt *Tabelle1*).

10.2 Einfügen von Zellen, Zeilen und Spalten

Mit Hilfe der Zugriffseigenschaften ENTIREROW und ENTIRECOLUMN ist zusammen mit der Methode INSERT das Einfügen von Zellen, Zeilen und Spalten möglich.

Beim Einfügen von Zellen kann mit Hilfe des Arguments SHIFT bestimmt werden, in welche Richtung im Verhältnis zur aktiven Zelle die übrigen Zellen verschoben werden sollen. Die Richtung wird über die integrierten Konstanten XLDOWN und XLTORIGHT gesteuert.

```
ActiveCell.EntireRow.Insert
ActiveCell.EntireColumn.Insert
ActiveCell.Insert Shift:= xlToRight
```

Die erste Anweisung fügt eine Zeile, die zweite eine Spalte ein. Es gilt die in Excel allgemein geltende Richtung – Zeilen werden oberhalb der aktiven, Spalten links von der aktiven eingefügt.
Die dritte Anweisung fügt eine Zelle ein, die übrigen Zellen werden nach rechts verschoben (Vorsicht: der Versatz geschieht nur in der Zeile der aktiven Zelle).

10.3 Zuweisen von Zellinhalten

Das schon mehrfach vorgestellte Zuweisen von Inhalten an Zellen wird mit Hilfe der VALUE – Eigenschaft realisiert. Diese Eigenschaft ist für ein RANGE – Objekt voreingestellt, d.h., wird bei Zuweisungen von Werten an Zellen die Eigenschaft nicht genannt, so wird VALUE automatisch angenommen.

```
ActiveCell.Value = 233
Range("B3").Value = "Hagen"
Range("B3:D3").Value = 75.25
Range("B3").Value = "31.12.99"
```

aber auch:

```
ActiveCell = 233
Range("B3") = "Hagen"
Range("B3:D3") = 75.25
Range("B3") = "31.12.99"
```

 Wird, wie in der letzten Anweisung ein Datum in der dort verwendeten Form übergeben, so wird es in die Zelle als Zeichenfolge übernommen. Für die Übergabe als Datumswert muß die Funktion CDATE benutzt werden:

```
Range("B3") = Cdate("31.12.99")
```

Das aktuelle Datum wird zugewiesen über:

```
Range("B3") = Date
```

die aktuelle Zeit mit:

```
Range("B3") = Time
```

Tips zu Praxislösungen:

1. Zuweisen von Werten in Abhängigkeit von Werten anderer Zellen.

```
Sub Zuweisung()
If IsNumeric(Range("B2").Value) Then
  ActiveCell.Value = Cells(3, ActiveCell.Column).Value * 0.75
Else
  MsgBox "Zelle B2 nicht numerisch"
End If
End Sub
```

2. Füllen eines Bereiches in Abhängigkeit von Werten in seinen Zellen

```
Sub Ersatz()
 For Each Wert In Range("B3:C4")
  If IsNumeric(Wert.Value) Then
    Wert.Value = Wert.Offset(0, 3).Value * Wert.Value
   Else
    Wert.Value = 0
  End If
 Next Wert
End Sub
```

Sind die Werte in B3:C4 numerisch werden sie mit den Werten der drei Spalten weiter rechts liegenden Zellen multipliziert. Die Ursprungswerte werden überschrieben. Für nicht numerische Werte soll das Ergebnis **0** sein.

Sollen die Nullen nicht im Tabellenblatt erscheinen, wird der Else – Zweig in der Form

```
Wert.Value = Empty
```

benutzt. Dies gleicht der Funktionskombination (EXTRAS / OPTIONEN / ANSICHT / NULLWERTE) von Excel.

10.4 Löschen von Zellinhalten und Zeilen

Zellen oder Zellbereiche können unterschiedliche Inhalte besitzen. Neben numerischen oder alphanumerischen Werten und Formeln können es Formatierungen (incl. Rahmen, Füllfarben, Muster) und Kommentare,. sein. Daher ist auch die Menge der Löschoptionen differerent:

CLEAR	Löscht im angegebenen Bereich alle Einträge und Formatierungen.
CLEARCONTENTS	Löscht im angegebenen Bereich alle Einträge
CLEARCOMMENTS	Löscht im angegebenen Bereich alle Kommentare.
CLEARFORMATS	Löscht im angegebenen Bereich alle Formate
DELETE	Löscht Zellen

```
Worksheets("Tabelle1").Range("B3").Clear
Worksheets("Tabelle1").Range("B3").ClearContents
Worksheets("Tabelle1").Range("B3").ClearFormats
```

Nach dem Löschen über die erste Anweisung sind die Zellen leer und besitzen alle das Standardformat. Die zweite Anweisung löscht nur die Inhalte (auch Formeln), die Formatierungen bleiben erhalten, die dritte dagegen löscht alle Formatierungen, die Inhalte bleiben erhalten.

 .Programmgesteuerte Löschaktionen besitzen keine Undo – Funktion. Auch die Methode Undo (Application Methode) gilt nur für die Rücknahme von Benutzeraktionen. Löschaktionen per Programm sind damit „unrückgängig“ !

 Bei der Verwendung von DELETE mit der Syntax

```
Objekt.Delete(Shift)
```

werden die gelöschten Zellen mit dem Inhalt der Zellen links oder darunter aufgefüllt. Für das Argument SHIFT die Konstanten XLUP und XLTOLEFT definiert.

DELETE ohne Argument verschiebt die unter der gelöschten Zelle liegenden Zellen nach oben (XLUP = default).

 Nur Zellen, die in der gleichen Zeile / Spalte wie die gelöschte liegen, werden bei DELETE verschoben, was oft ein „Chaos“ im Tabellenblatt erzeugt.

Für das Löschen kompletter Zeilen wird ebenfalls DELETE verwendet. Die Anweisungen

```
ActiveCell.EntireRow.Delete
ActiveCell.EntireColumn.Delete
```

Löschen die Zeile (1) oder Spalte (2) mit der aktiven Zelle.

 Sollen nur die Inhalte ganzer Zeilen oder Spalten, werden die schon beschriebenen Methoden CLEAR bzw. CLEARCONTENTS benutzt:

```
ActiveCell.EntireRow.Clear
ActiveCell.EntireColumn.ClearContents
```

10.5 Einfügen von Kommentaren

Zellen können mit Kommentaren versehen werden, die erklärende Texte enthalten. Diese erscheinen im Tabellenblatt, sobald die mit einem Kommentar versehene Zelle mit dem Mauszeiger berührt wird (Excel – Funktionskombination EINFÜGEN / KOMMENTAR).

Programmtechnisch kann es über das COMMENT – Objekt realisiert werden. Diesem Objekt sind entsprechende Eigenschaften und Methoden zugeordnet:

ADDCOMMENT	Methode zum Erzeugen eines Kommentars.
TEXT	Methode, die einem Kommentar – Objekt additiv oder ersetzend einen Text zuordnet.
VISIBLE	Eigenschaft für die Anzeige des Kommentartextes
DELETE	Methode zum Löschen des Kommentartextes
PREVIOUS	Methode, die das vorhergehende Comment – Objekt zurückgibt.
NEXT	Methode, die das nächste Comment – Objekt zurückgibt.

Eingefügt wird ein Kommentar mit einer Anweisung der Form:

```
Worksheets("Tabelle1").Range("B3").AddComment "Text des Kommentars"
```

Die eingefügten Kommentare sind in der Auflistung COMMENTS zusammengefaßt. Über die Eigenschaft Visible werden die Kommentare angezeigt oder ausgeblendet:

```
Worksheets("Tabelle1").Range("B3").Comment.Visible = True
Worksheets("Tabelle1").Range("B3").Comment.Visible = False
```

Die Methode Text erlaubt die nachträgliche Änderung des Kommentartextes. Die Syntax:

```
Objekt.Text(Text, Startposition, Overwrite)
```

mit den Argumenten:

TEXT	für den ergänzenden oder ersetzenden Kommentartext
STARTPOSITION	Position, ab der ersetzt oder eingefügt werden soll. Wenn nicht angegeben, wird vollständig ersetzt.
OVERWRITE	logisches Argument zur Bestimmung: Ersetzen =ja / nein

werden die Änderungen gesteuert.

Die Anweisungen:

```
Worksheets("Tabelle1").Range("B5").AddComment "Kein Eintrag !"
Range("B5").Comment.Text " neuer", 5, False
Range("B5").Comment.Visible = True
Range("B5").Comment.Visible = False
Range("B5").Comment.Text "Erfassen !"
```

Führen die folgenden Aktionen durch: (1) fügt einen Kommentar in die Zelle B5 ein, (2) fügt ab Position 5 des Kommentars eine Zeichenkette ein, (3) macht den Kommentar permanent sichtbar (kann im Tabellenblatt nicht per Mausklick ausgeblendet werden), (4) macht die Aktion von (3) rückgängig, (5) ersetzt den in (1) gesetzten Kommentartext durch einen neuen (STARTPOSITION nicht angegeben, bedeutet automatisch = 1, OVERWRITE nicht angegeben, bedeutet automatisch =TRUE !)

10.6 Benennen von Zellen und Zellbereichen

In EXCEL – Tabellen können über die Funktionskombination EINFÜGEN / NAMEN / FESTLEGEN einzelne Zellen oder Zellbereiche mit Namen versehen werden. Im VBA wird die Namenszuweisung über die für RANGE – Objekte definierte Eigenschaft NAME realisiert.

Die Anweisungen:

```
Range("B5").Name = "MwSt"
Range("C7").Name = "Netto"
Range("D1:E5").Name = "Endwerte"
```

definieren Namen für Zellen bzw. Zellbereiche.

Da Namen zu der für WORKSHEET- und WORKBOOK – Objekte zugeordneten Auflistung NAMES gehörende Objekte sind, kann für die Zuordnung eines Namens auch die ADD – Methode benutzt werden:

```
ActiveWorkbook.Names.Add "Ergebnis", "=Ergebnisse!$F$7"
```

 Zu beachten ist die Angabe des zweiten Parameters: das Gleichheitszeichen und die absolute Form der Zellbezüge müssen angegeben werden !

Bereichsnamen lassen sich nach der Zuweisung als Argumente von RANGE – Objekten benutzen:

```
Sub Testen()
    Range("B5").Name = "MwSt"
    Range("C7").Name = "Netto"
    Range("D1:E5").Name = "Endwerte"
    Worksheets("Ergebnisse").Range("MwSt").Value = 0.16
    Worksheets("Ergebnisse").Range("Endwerte").Value = 0
    Worksheets("Ergebnisse").Range("Endwerte").Value = _
    Range("MwSt").Value * Range("Netto").Value
End Sub
```

Die Bereichsnamen können gelöscht werden über eine Anweisung mit der Form:

```
Range("Endwerte").Name.Delete
```

 Die Funktionskombination EINFÜGEN / NAMEN / FESTLEGEN / LÖSCHEN der Excel – Oberfläche kann ebenfalls zu Löschen benutzt werden.

10.7 Suchen von Zellinhalten

Das Suchen nach bestimmten Zellen oder Zellbereichen ist oft nur über das Suchen nach bestimmten Inhalten der Zellen zu bewerkstelligen. Im VBA sind dafür mehrere für ein RANGE – Objekt definierte Methoden vorhanden, von denen zwei hier vorgestellt werden.

Die beiden vorgestellten Methoden besitzen eine Vielzahl von recht komplexen Argumenten, auf die hier nicht näher eingegangen wird – näheres siehe Online – Hilfe.
Die nachfolgenden Beispiele demonstrieren die einfachste Suche und das Ersetzen von Zelleninhalten.

FIND	Sucht eine Bereich nach Informationen durch und gibt im Erfolgsfall eine Range – Objekt zurück.
REPLACE	Sucht und ersetzt Zelleninhalte

 Find gibt ein Range – Objekt zurück, welches anschließend ausgewertet werden muß.

```
Wert1 = Range("B1:D5").Find("Summe:").Value
MsgBox Wert1
```

```
Wert1 = Range("B1:D5").Find("Summe:").Address
Range(Wert1).Value = 2222
```

```
Range("B1:D5").Replace "Summe2:", 2222
Range("B1:D5").Find("Summe:").Interior.Pattern = 2
```

Im ersten Beispiel wird im Bereich B1:D5 die Zelle mit dem String *Summe:* gesucht und ihr Inhalt (Value) in die Variable Wert1, die per MsgBox ausgegeben wird, übertragen.

Im zweiten Beispiel wird die Adresse der Zelle mit dem String *Summe:* ermittelt und der Variablen *Wert1* zugewiesen. Diese wird anschließend für die Zuweisung eines neuen Wertes an die Zelle benutzt.

Die erste Anweisung der dritten Beispielgruppe erledigt die im Beispiel 2 in zwei Anweisungen ausgeführte Wertänderung in einer Anweisung.
Die zweite Anweisung weist der Zelle mit dem gesuchten Inhalt ein Hintergrundmuster zu.

10.8 Suchen von Zelleninhalten über Schleifen

Die Suche nach Zelleninhalten, die in mehreren Zellen auftreten können ist am einfachsten über EACH – Schleifen möglich. Dabei können die Größer- / Kleiner – Operatoren, das Gleichheitszeichen oder der LIKE – Operator benutzt werden.

Die Suche über Schleifen hat noch einen zusätzlichen Vorteil – es ist möglich nicht nur nach Zellinhalten zu suchen, sondern auch nach Formatelementen (Näheres zu diesen Elementen –siehe Folgekapitel).

Für die Suche nach Zelleninhalten wird die Eigenschaft VALUE benutzt. Dabei ist zu beachten, daß die Inhalte unterschiedlich sein können – numerisch, alphanumerisch oder aus Formeln bestehend. Entsprechend muß in den Suchkonstrukten reagiert werden, um nicht unnötige Fehlermeldungen zu erzeugen.

Die Anweisungen des folgenden Beispiels sollen die Suche nach numerischen Werten demonstrieren, die ein bestimmtes Kriterium erfüllen (> 20). Im Positivfall soll der Zellenhintergrund farblich abgesetzt werden:

```
Sub Suche()
  For Each Zelle In Range("B2:D7")
   If IsNumeric(Zelle.Value) Then
     If Zelle.Value > 20 Then
       Zelle.Interior.ColorIndex = 5
     End If
    End If
  Next
End Sub
```

Da im angegebenen Bereich aber auch nichtnumerische Zelleninhalte stehen können, sollte vor der Prüfung auf den Wert eine zusätzliche Prüfung auf numerischen Inhalt (ISNUMERIC) eingebaut werden. Ohne dieser Prüfung läuft das Programm auf einen Fehler.

Bei der Suche nach alphanumerischen Inhalten, sollte beachtet werden, daß diese aus Klein-, Grossbuchstaben oder (im Regelfall) aus einem Mix von beiden bestehen können. Um hier Fehler zu vermeiden, sollte vor der Suche der Zelleninhalt „auf Gleichstand“ mit dem Suchbegriff gebracht werden:

```
Sub Suche()
  For Each Zelle In Range("B2:D7")
   If LCase(Zelle.Value) Like "wert*" Then
        Zelle.Borders.LineStyle = xlDouble
   End If
  Next
End Sub
```

Dieses Beispiel soll alle Zellen des angegebenen Bereichs dann mit einem doppelten Rahmen versehen, wenn der Inhalt eine aus Buchstaben *w e r t* bestehenden Zeichenkette am Anfang enthält. Über LCase wird vor dem Vergleich der Zelleninhalt in Kleinbuchstaben umgewandelt (die Zeichenkette in den Zellen kann also *wert, Wert, WERT WeRt*, usw. heißen). Das Sternchen hinter der Zeichenkette erlaubt beliebige Fortsetzungen der Kette in den Zellen.

Das nächste Beispiel such innerhalb des angegebenen Bereichs nach einer bestimmten Farbe des Hintergrunds (blau) und wenn gefunden, ordnet es der Zelle einen gepunkteten Rahmen, eine andere Hintergrundfarbe (gelb) und einen Wert (*Gesamt*) zu:

```
Sub Suche()
  For Each Zelle In Range("B2:D7")
   If Zelle.Interior.ColorIndex = 5 Then
        Zelle.Borders.LineStyle = xlDot
        Zelle.Interior.ColorIndex = 27
        Zelle.Value = "Gesamt:"
   End If
  Next
End Sub
```

10.9 Schriften, Rahmen, Farben

Die Zuordnung von Schriften, Rahmen und Farben an Zellen ist über dafür von VBA zur Verfügung gestellten Zugriffseigenschaften möglich. Die Palette dieser Eigenschaften ist recht umfangreich, eine Auswahl der gängigsten wird hier vorgestellt.

10.9.1 Zuordnung von Schriften

Für die Zuordnung von Schriften und Schriftattributen wird die Zugriffseigenschaft FONT benutzt. Sie gibt ein FONT – Objekt zurück, für welches u.a. die folgenden Eigenschaften definiert sind:

BOLD	wenn, =TRUE gesetzt, wird Fettschrift ausgegeben.
COLOR	Definition der Schriftfarbe über einen Farbwert einer Mischfarbe
COLORINDEX	Definition einer Schriftfarbe aus der Excel – Farbpalette (Wert 1 bis 56)
FONTSTYLE	Schriftstil des Zellinhalts
ITALIC	wenn = TRUE gesetzt - Ausgabe kursiver Schrift
NAME	Name der Schriftart
SIZE	Schriftgröße in Punkten (= 1/72 Zoll)
SUBSCRIPT	wenn = TRUE gesetzt – tiefgestellte Schrift
SUPERSCRIPT	wenn = TRUE gesetzt – hochgestellte Schrift
UNDERLINE	wenn = TRUE gesetzt – unterstrichene Schrift

Benutzt wird FONT in der allgemeinen Syntaxform:

```
Range.Font.Eigenschaft = Wert
```

beispielsweise:

```
Range("B3:C4").Font.Size = 18
```

In der Praxis werden oft gleichzeitig mehrere Schriftattribute zugeordnet, womit es sinnvoll ist, die Zuordnung in einer WITH – Struktur durchzuführen:

```
Sub Schrift()
With Worksheets("Tabelle1").Range("B1:B5").Font
      .Size = 14
      .Bold = True
      .Italic = True
      .Name = "Arial"
      .ColorIndex = 4
End With
End Sub
```

Die Eigenschaft FONT erlaubt nicht nur Zuweisungen von Schriftattributen, sondern auch deren Ermittlung (z.B. für evtl. Änderungen):

```
Sub Ersetzen()
  For Each Schrift In Range("B1:B5")
    If Schrift.Font.Size = 14 Then
      Schrift.Font.Size = 8
      Schrift.Font.Italic = False
    End If
  Next
End Sub
```

10.9.2 Zuordnung von Rahmen

Die Zuordnung von Rahmen ist über die Zugriffseigenschaft BORDERS oder die Methode BORDERAROUND möglich.

Die BORDERS – Eigenschaft , die Rahmen an einzelne Zellen oder Zellenbereiche zuordnen kann, hat zwei Syntaxvarianten:

```
Objekt.Borders
```

für die Zuordnung von Gesamtrahmen

```
Objekt.Borders(Index)
```

für die Zuordnung von Teilrahmen über den INDEXWERT mit den aus Konstanten bestehenden Argumenten XLTOP, XLBOTTOM, XLLEFT und XLRIGHT.

Für die Eigenschaft sind u.a. die Eigenschaften

COLOR	Rahmenfarbe als Mischwert
COLORINDEX	Rahmenfarbe aus der Excel – Palette (Wert 1 bis 56)
LINESTYLE	Rahmenart, durch die Konstanten XLCONTINUOUS, XLDASH, XLDOUBLE, XLDOT und XLNONE definiert (XLNONE entfernt den Rahmen).
WEIGHT	Rahmenstärke, durch die Konstanten XLHAIRLINE, XLMEDIUM, XLTHICK, XLTHIN definiert.

Wie schon bei der Zuweisung von Schriftattributen, ist es bei Rahmen möglich, einzelne Attribute zuzuordnen:

```
Range("B1:B5").Borders.LineStyle = xlDot
```

aber i.d.R. werden auch hier mehrere Attribute gleichzeitig zugeordnet (auch hier, wie bei den Schriftattributen) über WITH – Strukturen:

```
Sub Rahmen()
  With Range("B1:B5").Borders
    .LineStyle = xlContinuous
    .ColorIndex = 26
    .Weight = xlThin
  End With
End Sub
```

setzt Rahmen mit den definierten Attributen um jede Zelle des angegebenen Zellenbereichs.

```
Sub Rahmen()
  With Range("B1:B5").Borders(xlBottom)
    .LineStyle = xlDouble
    .ColorIndex = 24
  End With
End Sub
```

setzt einen doppelten Teilrahmen unten in jede Zelle des angegebenen Zellenbereichs.

Die Methode BORDERAROUND erstellt einen Rahmen um einen angegebenen Zellenbereich herum. Die im Bereich liegenden Zellen erhalten keine Einzelrahmen. Die Syntax entspricht der oben schon beschriebenen.

```
Range("B1:B5").BorderAround LineStyle:=xlDot, ColorIndex:=25
```

 Die Argumente LINESTYLE und WEIGHT dürfen nur alternativ verwendet werden, da sie miteinander unverträglich sind.

10.9.3 Zuordnung von Farben

Farben können an Schriften, Rahmen und Zellehintergrund zugeordnet werden. Die Zuordnung wird über die Eigenschaften COLOR oder COLORINDEX realisiert. Während die Eigenschaft COLORINDEX mit Indexwerten der EXCEL – Farbpaletten (Werte 1- 56) arbeitet und damit einfacher in der Anwendung ist, erfordert die COLOR – Eigenschaft einen Farbcode für die sich aus Grundfarben zusammensetzenden Mischfarben.

Das folgende, sehr einfach (und damit etwas umständlich) programmierte Beispiel zeigt eine mögliche Anwendung:

```
Sub Farbe1()
      Range("B1:B5").Interior.ColorIndex = 17
End Sub

Sub Farbe2()
  Range("B3").Activate
  Farbe = ActiveCell.Interior.ColorIndex
    If Farbe = 17 Then
      With ActiveCell
      .Interior.ColorIndex = 28
      .Offset(-1, 0).Interior.ColorIndex = xlNone
      .Offset(1, 0).Interior.ColorIndex = xlNone
      .Borders.LineStyle = xlDouble
      .Borders.ColorIndex = 25
      End With
    End If
End Sub
```

In der Sub FARBE1 wird einem Zellbereich eine Hintergrundfarbe zugeordnet.

In der Sub FARBE2 wird eine Zelle aktiviert und deren Hintergrundfarbe ermittelt. Der Indexwert wird der Variablen Farbe zugewiesen. Ist der Indexwert = 17, so wird die Hintergrundfarbe verändert, der Zelle ein Farbiger doppelter Rahmen zugeordnet und die Hintergrundfarbe der beiden darüber und darunter liegenden Zellen gelöscht.

Die COLOR – Eigenschaft arbeitet mit Mischfarben, die durch Codes für die Farben Rot, Grün und Blau definiert werden. Gearbeitet wird dabei mit dem RGB – Farbsystem, welches mit Wert zwischen 0 und 255 für die Grundfarben arbeitet. VBA besitzt für diesen Zweck eine eigene RGB – Funktion mit der allgemeinen Syntax:

```
RGB(Rot, Grün, Blau)
```

Die Farbe Rot kann beispielsweise zugeordnet werden mit:

```
RGB(255,0,0)
```

Die gemischten Farben können sowohl Hintergründen als auch Rahmen und Schriften zugeordnet werden, wie das nachfolgende Beispiel zeigt:

```
With ActiveCell
      .Interior.Color = RGB(125, 200, 20)
      .Offset(-1, 0).Interior.ColorIndex = xlNone
      .Offset(1, 0).Interior.ColorIndex = xlNone
      .Borders.LineStyle = xlContinuous
      .Borders.Color = RGB(30, 160, 120)
      .Font.Color = RGB(130, 0, 150)
    End With
```

Für die Grundfarben verfügt VBA über eine Auswahl an Farbkonstanten :

VBBLACK, VBBLUE, VBCYAN, VBGREEN, VBMAGENTA, VBRED, VBWHITE und VBYELLOW, die anstelle der RGB - Funktion eingesetzt werden können:

```
ActiveCell.Borders.Color = vbBlue
ActiveCell.Font.Color = vbYellow
```

Für die praktische Anwendung eine einfache Funktion, die Zellen mit definierter Hintergrundfarbe in einem Bereich zählt (z.B. durch die Excel – Funktionskombination FORMAT / BEDINGTE FORMATIERUNG eingefärbt):

```
Public Function IsFarbe(Bereich As Range, Farbe As Integer) As Integer
 Dim i As Integer, b As Variant
 For Each b In Bereich.Cells
   If b.Interior.ColorIndex = Farbe Then
    i = i + 1
   End If
 Next b
 IsFarbe = i
End Function
```

11 Aktionen auf Tabellenblättern

Die folgenden Beispiele demonstrieren einige Standardaktionen auf / mit Tabellenblättern, die in VBA – gesteuerten Abläufen öfters vorkommen. Sie bilden nicht die Gesamtheit aller möglicher Aktionen ab. Andere Alternativen sind bei Bedarf der Online – Hilfe zu entnehmen.

11.1 Einfügen neuer Tabellenblätter

Soll in einer aktiven Arbeitsmappe ein neues Tabellenblatt erstellt werden, gibt es dafür mehrere Möglichkeiten:

- Ein neues Tabellenblatt, das vor dem aktiven Blatt eingefügt werden soll, wird erstellt über:.

  ```
  ActiveWorkbook.Worksheets.Add
  ```

  Eine andere Alternative mit anschließender Benennung des neu eingefügten Tabellenblattes:

  ```
  Sheets.Add
  ActiveSheet.Name = "AA2"
  ```

- Das Einfügen mehrerer Tabellenblätter ist möglich über:

  ```
  Worksheets.Add Count:=2, Before:=Sheets(1)
  ```

  Hier werden 2 Tabellenblätter vor dem ersten Blatt der Arbeitsmappe eingefügt. Sie erhalten die Namen *Tabelle n+1* und *Tabelle n+2*, wobei *n* Index des letzten Tabellenblattes der Arbeitsmappe ist.

  ```
  Worksheets.Add Count:=3, Before:=Sheets("AA2")
  ```

  Diese Anweisung fügt 3 Tabellenblätter hinter dem Blatt mit dem Namen AA2 - zur deren Namen s.o.

11.2 Verschieben von Tabellenblättern

Die Reihenfolge der Tabellenblätter in einer Arbeitsmappe kann verändert werden. Die Tabellenblätter können durch die Anweisungen

```
Worksheets("Tabelle1").Move after:=Worksheets("Tabelle3")
Worksheets("Daten").Move before:=Worksheets("Ergebnisse")
```

verschoben werden.

Die erste Anweisung verschiebt die *Tabelle1* hinter die *Tabelle3*.

Die zweite Anweisung verschiebt die Tabelle *Daten* vor die Tabelle *Ergebnisse*

Die Anweisung

```
Worksheets("Tabelle9").Move After:=Worksheets(Worksheets.Count)
```

verschiebt das Tabellenblatt *Tabelle9* ans Ende des Indexbereichs der Tabellenblätter (Tabelle9 wird zur letzten Tabelle der Arbeitsmappe).

```
Worksheets.Add.Move After:=Worksheets(Worksheets.Count)
```

Die obige Anweisung ist ein „Sonderfall“ – sie fügt ein neues Tabellenblatt hinter dem aktiven Blatt und verschiebt es gleichzeitig ans Ende des Blattregisters.

11.3 Tabellenblätter aktivieren

Tabellenblätter können individuell aktiviert werden (das aktive Blatt gewechselt werden). Über eine Anweisung der Form:

```
Worksheets("Tabelle1").Activate
```

wird beispielsweise das Blatt *Tabelle1* aktiviert.

Eine andere Art des Aktivierens kann das Verbergen bzw. wieder sichtbar machen von Tabellenblättern sein. Tabellenblätter können „versteckt“ werden (wenn sie beispielsweise Daten enthalten, die nicht gesehen werden sollen). Dazu sind Anweisungen der folgenden Form nötig:

```
Worksheets(1).Visible = False
Worksheets("Werte").Visible = False
```

Die erste der beiden Anweisungen benutzt den Indexwert der Blattauflistung, die zweite arbeitet mit einem Blattnamen. In beiden Fällen wird das Tabellenblatt aus der Excel – Oberfläche ausgeblendet (unsichtbar). Der Zugriff auf die darin enthaltenen Daten ist jedoch weiterhin im vollen Umfang möglich.

Ausgeblendete Tabellenblätter werden mit:

```
Worksheets(1).Visible = true
Worksheets("Werte").Visible = true
```

wieder eingeblendet.

12 Dialoge (Teil II)

Neben den im Kapitel 8 vorgestellten Standarddialogen werden in Excel – Anwendungen andere Dialogformen benötigt, um Programmfunktionen aufrufen und ausführen zu können. Excel bietet dafür entweder mehrere Alternativen an, von denen zwei hier vorgestellt werden sollen – in Tabellenblättern plazierte Formularobjekte und spezielle VBA – Dialogobjekte.

12.1 Das Tabellenblatt als Formular

Tabellenblätter sollen eigentlich die für sie gedachte Aufgabe erfüllen – die Kalkulation. Sie können jedoch auch als Formulare benutzt werden. Für diese Aufgabe sollte ein Tabellenblatt allerdings erst vorbereitet werden. Dazu müssen aus dem Tabellenblatt alle Elemente entfernt werden, die solche Aktionen ermöglichen.

12.1.1 Vorbereitende Arbeiten

Dem Anwender soll ein Tabellenblatt präsentiert werden, in dem er keine Möglichkeit haben soll, sich frei in den Zellen zu bewegen oder in andere Tabellenblätter zu wechseln:

- **Die Gitternetzlinien** werden ausgeschaltet. Dieses ist zwar auch per Programm möglich, aber am einfachsten ist es, sie für das als Formular gedachte Blatt über die Funktionskombination
  EXTRAS / OPTIONEN / ANSICHT / FENSTEROPTIONEN / GITTERNETZLINIEN
  auszuschalten. Da Formularen oft ein farbiger Hintergrund zugewiesen wird, erübrigt sich dadurch das Ausschalten der Gitternetzlinien, da die zugewiesene Farbe sie sowieso überdeckt. Die Farbzuweisung erfolgt auf dem Excel – üblichen Weg – markieren und Farbe zuweisen.

   Der farbige Bereich sollte über den sichtbaren Bereich des Tabellenblattes hinausgehen. Wenn die Applikation auf einem Rechner laufen soll, dessen Bildschirm eine andere Auflösung hat als der des Entwicklungsrechners – vorher austesten (auch in Bezug auf Plazierung des Steuerelemente – siehe weiter im Text).
- **Die Bildlaufleisten** sowie **Zeilen- und Spaltenköpfe** werden ebenfalls über die Funktionskombination EXTRAS / OPTIONEN / ANSICHT / FENSTEROPTIONEN ausgeblendet.
- **Die Tabellenblattregister** (Arbeitsmappenregister) können ebenfalls ausgeblendet werden (gleiche Funktionskombination) allerdings nur für die gesamte Arbeitsmappe. Der Wechsel von Tabellenblättern ist dann über eigene Schaltflächen innerhalb der als Formular zu benutzenden Blattes möglich.
- **Die Symbolleisten** können ebenfalls ausgeblendet werden (Funktionskombination ANSICHT / SYMBOLLEISTEN).
- **Die Menüleiste** kann über die Funktionskombination EXTRAS / ANPASSEN entweder auf unbedingt benötigte Befehle reduziert oder gänzlich entfernt werden, allerdings Vorsicht: wird die Leiste komplett entfernt, so wird die Bedienung der Excel – Oberfläche während der Anwendungskonzeption kaum mehr möglich sein.

 Vorsicht bei der Durchführung der hier genannten „vorbereitenden“ Arbeiten – man sollte sie erst dann vollständig durchführen, wenn das Formular endgültig realisiert ist, damit man sich selbst beim Arbeiten nicht behindert.

 Das so vorbereitete Formular bleibt immer noch ein Tabellenblatt. Es besitzt u.a. noch die Fähigkeit, Rahmen und Linien aufzunehmen. Damit ist die optische Hervorhebung bestimmter Bereiche des Formulars möglich.

 Soll die Anwendung mehrere Formulare mit gleicher Standardform enthalten, kann ein vorbereitetes Formular kopiert werden. Dazu Blattregister anklicken und bei gedrückter STRG – Taste an die gewünschte Position verschieben.

12.2 Formularsteuerelemente

Die Formularsteuerelemente, enthalten in der Symbolleiste Formular sind nur mit Einschränkungen dafür geeignet, in einem steuernden Dialog verwendet zu werden, weil sie eigentlich darauf ausgerichtet sind, ohne VBA – Code auszukommen. Ihre Programmierbarkeit beschränkt sich in der Praxis auf die Zuweisung von Makros (Subs). Außer einer kleinen Beispielanwendung mit diesen Steuerelementen werden deshalb in dieser Unterlage hauptsächlich die Steuerelemente der Toolbox behandelt.

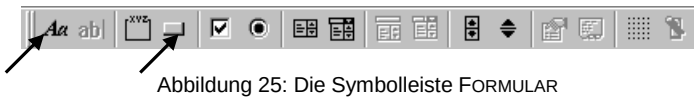


Abbildung 25: Die Symbolleiste FORMULAR

Zwei der Steuerelemente der Symbolleiste FORMULAR können jedoch durchaus für die Gestaltung von Formularen (insbesondere von „Eingangsformularen“), trotz der oben genannten Einschränkung, gut benutzt werden – die Elemente BEZEICHNUNGSFELD und SCHALTFLÄCHE.

Die Anwendung dieser Elemente in einem Formular zeigt das Beispiel einer sehr einfachen Anwendung im Kapitel 12.4.

12.3 Ein Tabellenformular öffnen

Ein im Tabellenblatt erzeugtes Formular sollte nach dem Öffnen der Arbeitsmappe aktiv sein.

Eine Arbeitsmappe zeigt als aktives Tabellenblatt immer das Blatt, welches beim Verlassen der Arbeitsmappe aktiv war. Direkt kann per Programm ein Tabellenblatt geöffnet werden über die Anweisungen:

```
Worksheets(Tabellenname).Activate
Worksheets(Index).Activate
```

Index ist der Positionsindex des Tabellenblattes in der Arbeitsmappe. Sicherer ist es, den Blattnamen zu verwenden, da sich die Reihenfolge der Tabellenblätter zur Laufzeit eines Programms ändern kann und damit der angegebene Indexwert auf ein falsches Blatt zeigen kann.

```
Worksheets("Startmenu").Activate
Worksheets(1).Activate
```

Zur Sicherheit sollte der Aufruf um den Namen des Arbeitsmappe erweitert werden, da man nicht immer davon ausgehen kann, daß das zu aktivierende Tabellenblatt in der gerade aktiven Arbeitsmappe liegt:

```
Workbooks("Mappe5").Worksheets("Startmenu").Activate
Workbooks("Mappe5").Worksheets(1).Activate
Workbooks("Kosten.xls").Worksheets("Startmenu").Activate
```

Ist die benötigte Arbeitsmappe nicht geöffnet, kann sie mit

```
Workbooks.Open("Kosten.xls")
```

im aktuellen Ordner geöffnet werden. Anschließend wird das gewünschte Tabellenblatt mit dem Formular aktiviert.

Befindet sich die Arbeitsmappe in einem anderen Ordner, sollte der komplette Pfad angegeben werden:

```
Workbooks.Open("D:\Bilanz\Abt1\Kosten.xls")
```

Geschlossen wird eine Arbeitsmappe mit einem Befehl der Form:

```
Workbooks("Kosten.xls").Close
```

Soll das Schließen ohne Abfrage nach dem Speichern eventueller Änderungen erfolgen, wird die Anweisung erweitert:

```
Workbooks("Kosten.xls").Close SaveChanges := True
Workbooks("Kosten.xls").Close SaveChanges := False
```

12.4 Beispielanwendung 1

Im folgenden Programmbeispiel soll ein Eingangsformular in einem Tabellenblatt mit Hilfe der Elemente SCHALTFLÄCHE der Symbolleiste FORMULAR den Ablauf einer einfachen Anwendung mit bestimmten Aufgaben steuern.

In einem vorbereiteten Tabellenblatt, in dem über die Funktionskombination EXTRAS / OPTIONEN die Laufleisten, die Eingabezeile, das Gitternetz, die Symbolleisten und der Tabellenindex entfernt und die Zellen mit einem farbigen Hintergrund belegt wurden, sollen die folgenden Aufgaben erfüllt werden:

- Im Blatt sollen im oberen linken Bereich in den Spalten A und B Artikelnamen und Nettopreise per Datenmaske erfasst werden können.
- Die Artikel sollen einem bestimmten Kunden zugeordnet sein. Der Name dieses Kunden soll per INPUTBOX erfasst und in das Tabellenblatt eingetragen werden können.
- Nach der Erfassung sollen in der Spalte C die Bruttopreise mit einem vorgegebenen MwSt-Satz ausgerechnet werden, im unteren rechten Teil des Datenbereichs sollen die Gesamtsumme und der MwSt-Anteil ausgerechnet werden.
- Sowohl die Netto- als auch die Brutto-Preise, die Gesamtsumme und der Anteil der MwSt am Gesamtpreis sollen im Währungsformat erscheinen, die Berechnungsformeln sollen per Programm in die entsprechenden Zellen eingesetzt werden.
- Der gültige MwSt-Satz soll über eine INPUTBOX eingegeben und in eine feste Zelle des Blattes eingesetzt werden können.
- Es soll die Möglichkeit bestehen, über eine Datenmaske Datenkorrekturen vorzunehmen, allerdings in einem eigenen Korrekturblatt. Nach der Korrektur sollen die Daten in das Originalblatt kopiert werden und das Korrekturblatt soll gelöscht werden.
- Bei Bedarf soll ein Hilfetext eingeblendet werden können. Dieser Text soll auf einem eigenen Tabellenblatt in einem Textfeld stehen (die einfachste Lösung). Dieses Blatt soll nach dem Verlassen ausgeblendet werden.
- Für einen Kundenwechsel soll es die Möglichkeit geben, die Daten zu löschen. Die Beschriftungen sollen im Blatt verbleiben.
- Die Anwendung soll über Schaltflächen gesteuert werden.
- Aus Gründen der Einfachheit soll für dieses Beispiel der komplette Programmcode in einem Modul, unterteilt in einzelne Sub's, untergebracht werden.

Das für die Aufnahme der Daten „präparierte“ und mit Schaltflächen versehene Tabellenblatt hat die folgende Form:

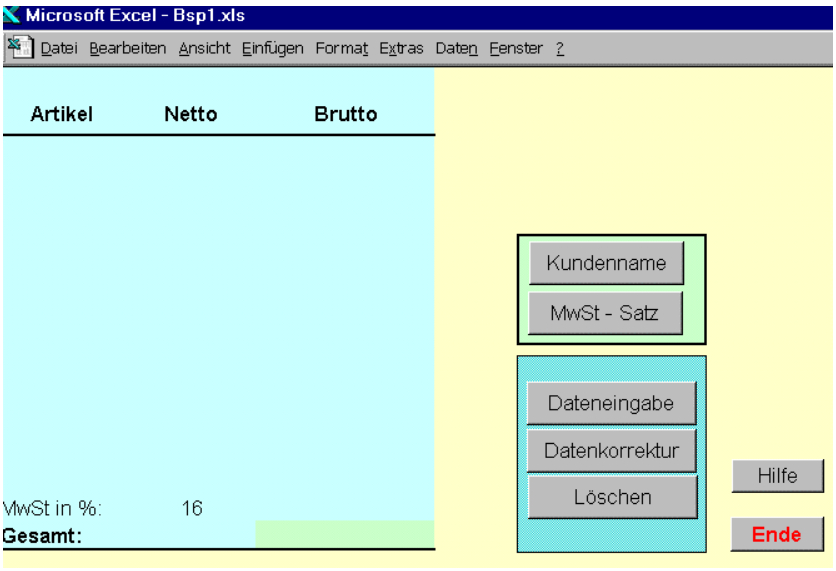


Abbildung 26: Tabellenblatt des Beispielprogramms

Der vollständige Programmcode ist im Anhang zu sehen. Die einzelnen Sub's wurden in einer möglichst einfachen Form geschrieben. Manche Teillösungen sind sicher nicht die elegantesten, aber dafür recht übersichtlich und nachvollziehbar gestaltet. Die einzelnen Sub's und ihre Funktion im Gesamtcode werden hier Schrittweise erläutert.

Der Programmcode beginnt mit einer AUTO_OPEN – Prozedur, die beim Öffnen der Arbeitsmappe automatisch ausgeführt wird[5]:

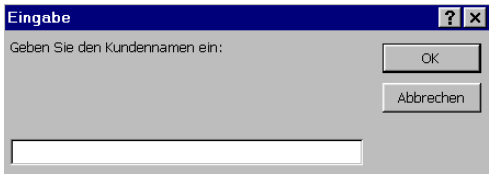
```
Sub Auto_open()
  Worksheets("Hilfe").Visible = False
  Sheets("Auswertung").Select
  Range("A2").Select
End Sub
```

Sie sorgt dafür, daß beim Öffnen der Arbeitsmappe das Eingangsblatt *Auswertung* geöffnet wird und das evtl. noch sichtbare Blatt *Hilfe* „versteckt" wird. Die Zelle *A2* (Beginn des Datenbereichs) wird markiert.

Ausgehend von der Situation – neuer Kunde, neue Artikelliste – wird über eine der Schaltfläche *Kundenname* zugeordnete Prozedur:

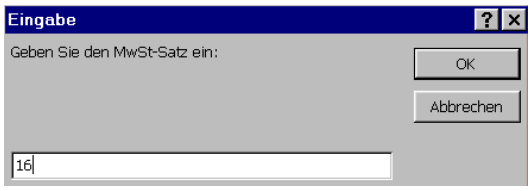
```
'====================================================
'     Eintrag Kundenname
'====================================================
Sub Kunde()
    Range("Auswertung!E1") = "Kunde: " & Application.InputBox( _
    prompt:="Geben Sie den Kundennamen ein:", Type:=2)
    Range("A2").Select
End Sub
```

in die Zelle E1 der über eine INPUTBOX einzugebende *Kundenname*, verknüpft mit der Zeichenkette *Kunde:* eingetragen:



Über ein ähnliches Verfahren wird über die Schaltfläche *MwSt-Satz* in die Zelle B15 des Blattes der ganzzahlig einzugebende MwSt – Satz eingetragen:

```
'====================================================
'     Eintrag MwSt
'====================================================
Sub Steuer()
    Range("Auswertung!B15") = Application.InputBox( _
    prompt:="Geben Sie den MwSt-Satz ein:", Type:=1)
    Range("A2").Select
End Sub
```



Der Eintrag der Daten ist über die Schaltfläche Dateneingabe möglich. Dieser Schaltfläche ist die folgende Prozedur zugeordnet:

[5] Zu den Prozeduren (Makros) AUTO_OPEN und AUTO_CLOSE gibt es in der EXCEL97 - Version zusätzlich die Ereignisprozeduren WORKBOOK_OPEN und WORKBOOK_CLOSE die dem Objekt WORKBOOK zugeordnet sind. AUTO_OPEN und AUTO_CLOSE werden aus Gründen der Rückwärtkompatibilität verwendet. Der Prozedurkörper bleibt in beiden Fällen identisch.

```
'=====================================================
'  Dateneintrag
'=====================================================
Sub Eintrag()

   ActiveWindow.DisplayZeros = False
   Range("B2:B13").Select
   Selection.Style = "Currency"
   Range("C2:C12").Select
    With Selection
      .Formula = "=B2+B2*$B$15/100"
      .Style = "Currency"
    End With
    Range("A2").Select
    Application.DisplayAlerts = False
     ActiveSheet.ShowDataForm
    Application.DisplayAlerts = True
    Range("C15").Formula = "=sum(C2:C13)*$B$15/100"
    Range("C16").Formula = "=sum(C2:C13)"
End Sub
```

Diese Prozedur unterdrückt die Anzeige von Nullwerten im Tabellenblatt, besetzt benötigte Zellen der Spalten B und C mit dem Währungsformat und trägt in die berechnenden Zellen der Spalte C die entsprechenden Formeln für die Berechnung der Bruttobeträge ein. Anschließend markiert sie die Zelle A2 und öffnet die Datenmaske. Da der Datenbereich bis dahin keine Daten, sondern nur die Überschriften enthält, würde an dieser Stelle eine Excel – Fehlermeldung (Info – Meldung) erscheinen. Diese wird vor dem Aufruf der Datenmaske mit DISPLAYALERTS = FALSE unterdrückt. Da dieses Verfahren aber bis zum Ende der Anwendung Meldungen unterdrückt, sollte dafür gesorgt werden, daß es zum geeigneten Zeitpunkt rückgängig gemacht wird. Im Beispiel geschieht es unmittelbar nach dem Verlassen der Datenmaske (DISPLAYALERTS = TRUE).

✘.Für die Dauer der Arbeit mit der Datenmaske wird der Ablauf der Prozedur gestoppt, d.h. die nachfolgenden Anweisungen werden erst nach dem Verlassen der Datenmaske ausgeführt.

Entsprechend dem obigen Hinweis werden also erst nach dem Ende des Dateneintrags in den Zellen C15 und C16 die Formeln eingetragen und erst dann wird gerechnet.

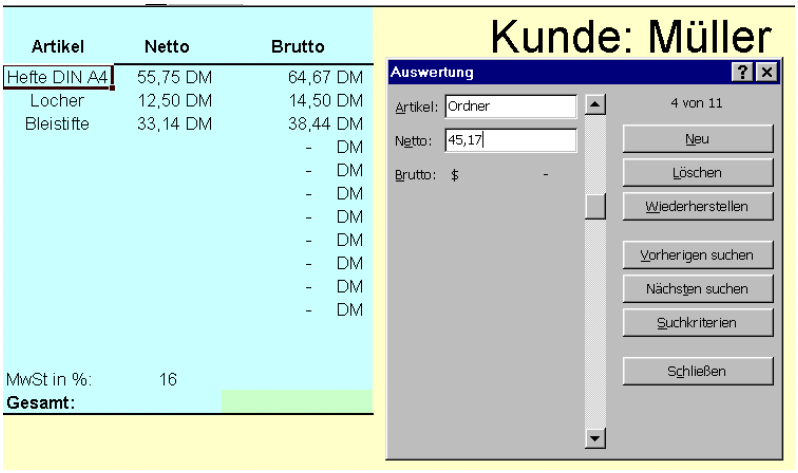


Abbildung 27: Formularblatt während der Datenerfassung

Nach dem Erfassen (Abschluß über die Schaltfläche SCHLIEßEN der Datenmaske) wird die Datenmaske ausgeblendet und der Datenbereich vollständig berechnet:

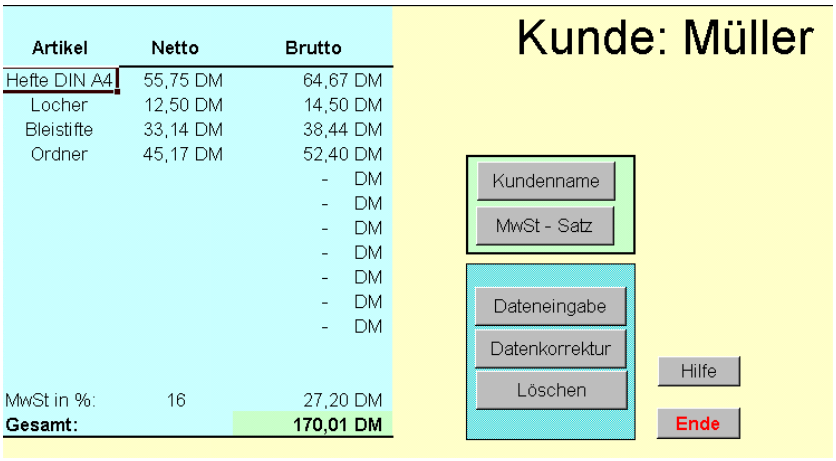


Abbildung 28: Formularblatt nach Abschluß des Erfassungsvorganges

Für das Löschen der Daten (Schaltfläche Löschen) werden zuerst die Datenbereiche (Artikeldaten, MwSt – Anteil und Gesamtpreis) ausgewählt:

```
'=====================================================
' Löschen Daten
'=====================================================
Sub Lösch_Daten()
Worksheets("Auswertung").Activate
Set r1 = Range(Cells(2, 1), Cells(13, 3))
Set r2 = Range(Cells(15, 3), Cells(16, 3))
Set MehrBlockBereich = Union(r1, r2)
MehrBlockBereich.Select
Selection.ClearContents
Range("E1").ClearContents
Range("A2").Select
End Sub
```

Da es sich um zwei nicht zusammenhängende Zellbereich handelt, werden sie zuerst über RANGE ermittelt und anschließend mit UNION zu einem Markierungsbereich vereinigt, dessen Inhalt mit ClearContents gelöscht wird.

Der Kundenname wird aus der Zelle E1 in einer eigenen Anweisung gelöscht. Als Vorbereitung zur einer neuen Dateneingabe wird am Ende die Zelle A2 aktiviert. Damit erhält das Formularblatt die Form aus der Abbildung 26 (plus aktivierte Zelle A2).

Die Datenkorrektur soll laut Vorgabe in einem eigenen Tabellenblatt durchgeführt werden. Dazu wird dieses zuerst angelegt und *Korrektur* genannt:

```
'=====================================================
'        Datenkorrektur
'=====================================================
Sub Korrektur()
    Sheets.Add
    ActiveSheet.Name = "Korrektur"
```

Aus dem Datenbereich im Blatt *Auswertung* werden die Zellen A1:B12 herauskopiert und in das Blatt *Korrektur* hineinkopiert:

```
    Sheets("Auswertung").Select
    Range("A1:B12").Select
    Selection.Copy
    Sheets("Korrektur").Select
    ActiveSheet.Paste
```

Der nach dem Kopieren im Blatt verbleibende Laufrahmen wird abgeschaltet, das Netzgitter sowie Zeilen- und Spaltenüberschriften werden ausgeblendet:

```
    Application.CutCopyMode = False
    ActiveWindow.DisplayGridlines = False
    ActiveWindow.DisplayHeadings = False
```

Nachließend wird in dem kopierten Datenbereich die letzte beschriebene Zelle der Spalte A der letzte eingetragene Artikel) ermittelt, um die Datenkopie auf den tatsächlich beschriebenen Datenbereich beschränken zu können. Die Zeilen und Spaltenindizes dieser Zelle werden für spätere Verwendung den Variablen A und B zugewiesen:

```
Range("A1").Select
  ActiveSheet.Cells(Rows.Count, ActiveCell.Column).Select
    If IsEmpty(ActiveCell) Then
      ActiveCell.End(xlUp).Select
      A = ActiveCell.Row
      B = ActiveCell.Column
    End If
```

Der übriggebliebene kopierte (leere) Datenbereich wird der Formate beraubt (Formeln und Farbe) und anschließend die Datenmaske für die Korrektur aufgerufen. Da hier der zu korrigierende Datenbereich schon gefüllt ist, muß keine evtl. Fehlermeldung erwartet und daher auch nicht, wie im Falle der Datenerfassung, unterdrückt werden (der Programmablauf wird für die Dauer der Arbeit mit der Datenmaske unterbrochen):

```
Range(Cells(A + 1, B), Cells(A + 9, B + 1)).ClearFormats
Range("A2").Select
ActiveSheet.ShowDataForm
```

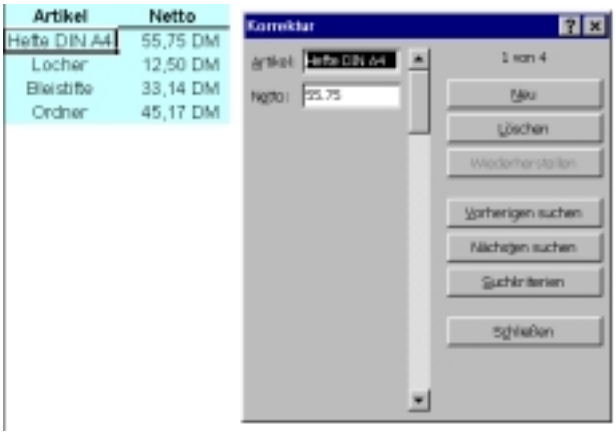


Abbildung 29: Datenkorrektur (nur eingetragene Daten sichtbar)

Nach Abschluß der Korrektur (Schaltfläche Schließen der Datenmaske) wird mit Hilfe der o.g. Variablen A und B der Datenbereich aus dem Korrekturblatt in das Originalblatt kopiert:

```
Range(Cells(A, B), Cells(1, B + 1)).Select
    Application.CutCopyMode = False
    Selection.Copy
    Sheets("Auswertung").Select
    Range("A1").Select
    ActiveSheet.Paste
```

Das Korrekturblatt wird gelöscht (die Systemmeldungen werden davor in schon oben beschriebener Art abgeschaltet, danach wieder eingeschaltet) und im verbleibenden Blatt Auswertung wird als rein optische Hervorhebung für den Anwender die Zelle C16 (Gesamtsumme) aktiviert:

```
    Sheets("Korrektur").Select
    Application.DisplayAlerts = False
    ActiveWindow.SelectedSheets.Delete
    Application.DisplayAlerts = True
    Range("C16").Select
End Sub
```

Für die „Online – Hilfe“ (hier absichtlich in Anführungsstrichen geschrieben, weil es in dieser Form nur eine beabsichtigte „Schnell – Notlösung“ ist) wurde ein eigenes Tabellenblatt mit einem Textfeld als Inhalt konzipiert. Dieses Tabellenblatt bleib solange unsichtbar, bis die Schaltfläche Hilfe betätigt wird:

```
'=====================================================
' Hilfe
'=====================================================
Sub Hilfe()
Worksheets("Hilfe").Visible = True
Worksheets("Hilfe").Select
Range("H20").Select
End Sub
```

Das Aktivieren der Zelle H20 im Blatt der Hilfe ist nur ein Trick, um die Markierung unter einer über dieser Zelle plazierten Schaltfläche zu verstecken. Das Problem ist auch anders lösbar, hier wurde es auf dem denkbar einfachsten Wege (sozusagen „zu Fuß“) gelöst:

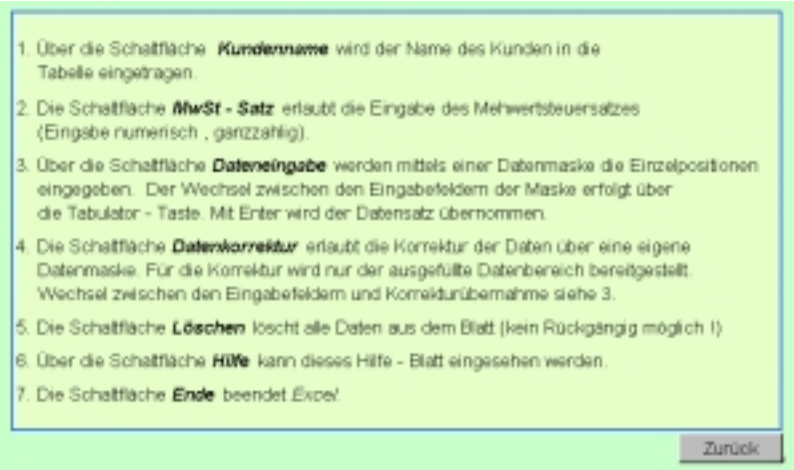


Abbildung 30: Inhalt des Blattes *Hilfe*

Über die Schaltfläche Zurück kann das Blatt der Hilfe verlassen werden. Mit dem Verlassen wird das Blatt aus der Anwendung ausgeblendet:

```
'=====================================================
' Hilfe verlassen
'=====================================================
Sub Hilfe_Aus()
Worksheets("Auswertung").Select
Worksheets("Hilfe").Visible = False
End Sub
```

Über die Schaltfläche *Ende* wird EXCEL und damit die Anwendung verlassen:

```
'=====================================================
' Beenden
'=====================================================
Sub Ende()
  Application.Quit
End Sub
```

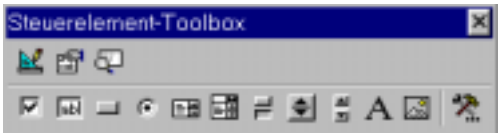
Wenn beim Aufruf der QUIT - Methode eine ungespeicherte Arbeitsmappe geöffnet ist, gibt EXCEL mit einem Dialogfeld die Möglichkeit, die Änderungen zu speichern.

➡ Das Dialogfeld erscheint nicht, wenn vor dem Aufruf der QUIT – Methode, wie oben mehrfach demonstriert, die DISPLAYALERTS-Eigenschaft auf FALSE gesetzt wurde. In diesem Fall zeigt EXCEL das Dialogfeld auch dann nicht an, wenn das Programm beendet wird und noch ungespeicherte Arbeitsmappen existieren, was zur Folge hat, daß, die Änderungen in den Arbeitsmappen verlorengehen.

➡ Die hier vorgestellte Lösung der Aufgabe läßt sich mit fortgeschrittenen Mitteln wesentlich eleganter darstellen. Da es sich jedoch hier um eine Einführung handelt und die „erwachsenen“ Techniken in dieser Unterlage nicht vorgestellt werden, wurde bei der Lösung zu den einfachsten Mitteln gegriffen, bei denen hoffentlich auch ein Einsteiger noch „durchblickt“, die der Profi jedoch vielleicht nur müde belächelt.

12.5 Steuerelemente aus der Toolbox

Die über die Funktionskombination ANSICHT / SYMBOLLEISTEN / STEUERELEMENTE-TOOLBOX aufzurufende Symboleiste:



enthält eine Auswahl ähnlicher, aus fast allen Windows – Anwendungen bekannter, Steuerelemente, die jedoch anders per Programm zu behandeln sind und die Gestaltung zum Teil sehr komplexer Oberflächen gestatten.

12.5.1 Einfügen der Elemente ins Tabellenblatt

Die Elemente werden in der Symbolleiste angeklickt und bei gedrückter linker Maustaste ins Tabellenblatt eingetragen (Rahmen zeichnen, nach dem Loslassen der Maustaste wird das Steuerelement in der eingezeichneten Größe angezeigt).

Die Bedeutung der Symbole (untere Reihe in der obigen Abbildung, von links nach rechts):

Kontrollfeld (CHECKBOX)	Kann die Zustände aktiv / inaktiv annehmen. Für die Aufnahme von Optionen zu verwenden. Mehrere Kontrollfelder in einem Dialog sind voneinander unabhängig.
Textfeld (TEXTBOX)	Zur Aufnahme von Eingaben während der Laufzeit eines Programms.
Schaltfläche (COMMANDBUTTON)	Schalterschaltfläche
Optionsfeld (OPTIONBUTTON)	Ähnlich der CHECKBOX . Kann allerdings zu Gruppen aus mehreren Elementen zusammengefaßt werden, in denen nur ein Element aktiv sein muß. Wird ein Element aktiviert, deaktiviert die Aktion ein anderes.
Listenfeld (LISTBOX)	Definition von Auswahllisten.
Kombinationsfeld (COMBOBOX)	Kombination von Listenfeld und Textfeld. Aufwendig in der Anwendung.
Umschaltfläche (TOGGLEBUTTON)	Wie Schaltfläche, jedoch zur Benutzung als Auswahl von mehreren alternativen Schaltflächen (aktiv = abgesenkt).
Drehfeld (SPINBUTTON)	Für die Eingabe numerischer Werte über Inkrement- und Dekrementschalter.
Laufleiste (SCROLLBAR)	Für die Eingabe numerischer Werte über Inkrement- und Dekrementschalter, Verschieben einer Schaltfläche oder Anklicken der Innenfläche.
Bezeichnungsfeld (LABEL)	Zur Erzeugung von Textobjekten für Hinweise oder Beschriftungen.
Bildfeld (PICTURE)	Feld zur Aufnahme einer Grafik.
Sonstige Elemente	Zum Einblenden anderer Elementen - Symbole.

12.5.2 Eigenschaften der Elemente

Die Steuerelemente der Toolbox können mit wesentlich mehr Eigenschaften ausgestattet werden als die der Formularleiste. Die Eigenschaften können per Dialogfenster oder direkt im Programm geändert werden.

Der Eigenschaften – Dialog wird aufgerufen, indem aus dem nach dem Klick mit der rechten Maustaste auf ein Dialogelement erscheinenden Auswahlmenu die Position Eigenschaften gewählt wird.

Die Listen der daraufhin erscheinenden Eigenschaften von Steuerelementen sind unterschiedlich lang. Einige der Eigenschaften (Größe, Position) sind einfacher per Mausklick einzustellen, andere sollten über das Dialogfeld definiert werden.

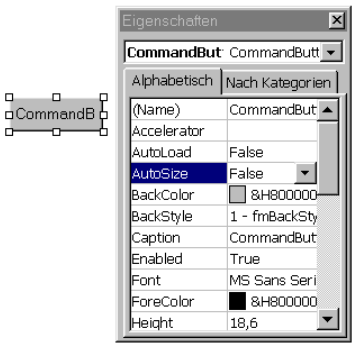


Abbildung 31: Dialog *Eigenschaften*
am Beispiel einer Schaltfläche

Die Eigenschaften werden durch Einträge bzw. Auswahlen in der rechten Spalte des Eigenschafts – Dialogs verändert. Endgültig zugewiesen wird die Eigenschaft erst nach den Verlassen der ausgewählten Zeile des Dialogfeldes.

Sind im Formular meherer Steuerelemente eingefügt, können deren Eigenschaften im Eigenschafts – Dialog in beliebiger Reihenfolge eingestellt werden. Da das Fenster mit den Eigenscahften nicht modal ist, reicht es aus, es für ein Steuerelement zu öffnen und anschließend durch Anklicken gewünschter anderer Elemente in deren Eigenschaftslisten zu wechseln.

Die Steuerelemente weisen eine Reihe gemeinsamer Eigenschaften, Methoden und Ereignisse auf, die hier vorab vorgestellt werden sollen. Sie können (neben vielen anderen) auch im Objektkatalog beim jeweiligen Control – Objekt eingesehen werden. Diese Eigenschaften können sowohl über den Eigenschafts – Dialog, als auch insbesondere zur Laufzeit eines Programms zugewiesen bzw. verändert werden.

Gemeinsame Eigenschaften	
NAME	Name des Steuerelements. Muß eindeutig sein Ohne Namen läßt sich das Steuerelement nicht im Programm verwenden. Excel vergibt automatisch einen Namen, sobald ein Element im Dialogformular eingesetzt wird. Der Name kann verändert werden, um aussagefähige individuelle Namen benutzen zu können.
CAPTION	Eigenschaft für Schalter, Options- und Kontrollfelder. Steht für die Beschriftung der Steuerelemente.
TEXT	Eigenschaft, die den Inhalt von Steuerelementen steuert, z.B. Inhalt von Textfeldern.
VALUE	Gibt den Zustand oder Inhalt eines angegebenen Steuerelements an, z.B. den Inhalt eines Textfeldes oder den Zustand einer Schaltfläche oder Optionsfeldes usw.
CANCEL	= TRUE, wenn das Steuerelement durch Esc verlassen werden kann. Wird normalerweise verwendet, damit die nicht abgeschlossene Änderungen abgebrochen werden können und den vorhergehenden Zustand eines Formulars wieder hergestellt werden kann.
CONTROLTIPTEXT	Gibt den Text an, der angezeigt wird, wenn der Benutzer den Mauszeiger eine Weile über ein Steuerelement hält, ohne zu klicken.
CONTROLSOURCE	Stellt die Verbindung zu einer Zelle im Tabellenblatt her. Bezeichnet die Datenposition, mit der die VALUE - Eigenschaft eines Steuerelements festgesetzt oder gespeichert werden kann.
DEFAULT	= TRUE, wenn das Element durch ENTER ausgewählt werden kann: Legt die Standardbefehlsschaltfläche eines Formulars fest.

LINKEDCELL	Bildet die Verbindung zwischen der Eigenschaft VALUE bzw. TEXT eines Elements und einer Zelle im Tabellenblatt. Ein Eintrag in Element wird an die über diese Eigenschaft zugeordnete Zelle weitergegeben und umgekehrt.
ROWSOURCE	Stellt die Verbindung zu einem Zellbereich her. Gibt die Quelle an, die eine Liste für ein Kombinationsfeld-Steuerelement (COMBOBOX) oder Listenfeld-Steuerelement (LISTBOX) zur Verfügung stellt.
TABINDEX	Gibt die Position eines einzelnen Objekts in der Aktivierreihenfolge des Formulars an.
TABSTOP	= TRUE, wenn das Element mit der Tab – Taste ausgewählt werden kann. Zeigt an, ob ein Objekt den Fokus erhalten kann, wenn die TAB - Taste drückt wird.
VISIBLE	= TRUE, wenn das Element sichtbar werden soll.
Gemeinsame Eigenschaften für die Stilbestimmung	
BACKCOLOR	Bestimmt die Hintergrundfarbe eines Steuerelements (für Text-, Listen-, Options- und Kontrollfelder).
BORDERCOLOR	Bestimmt die Rahmenfarbe (für Text-, Listen-, Options- und Kontrollfelder).
BORDERSTYLE	Betsimmt den Rahmentyp (für Text-, Listen-, Options- und Kontrollfelder).
FONT	Schriftart und Schriftattribute.
FORECOLOR	Farbe des Fordergrundes des Elements.
SHADOW	Darstellung eines Schattens.
SPECIALEFFEKT	Zuweisung von 2D / 3D – Effekten.
Gemeinsame Methoden	
SETFOCUS	Setzt den Eingabefokus auf das Element.

12.5.3 Das Bezeichnungsfeld (Label)

Das Bezeichnungsfeld wird zur Beschriftung eines Dialogs benutzt. Es wird neben Steuerelementen plaziert, um sie zu Beschriften bzw Hinweise zu deren Benutzung geben zu können.

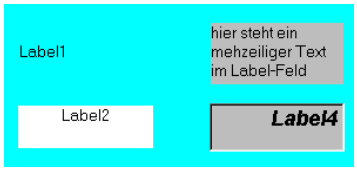


Abbildung 32: Unterschiedliche Formen des LABEL -Feldes

Wichtigste Eigenschaften:

- Der darzustellende Text wird über die Eigenschaft CAPTION definiert.
- Bei mehrzeiligen Texten muß die Eigenschaft WORDWRAP auf TRUE gesetzt werden.
- Die Textausrichtung wird über die Eigenschaft TEXTALIGN geregelt.
- Soll sich die Größe des Feldes an die Länge des Textes anpassen, muß AUTOSIZE auf TRUE gesetzt werden.
- Schriftart und –farbe werden über FONT, BACKCOLOR bzw. FORECOLOR eingestellt.
- Die Art der Umrandung kann über BORDERSTYLE und BORDERCOLOR definiert werden.
- 3D – Effekte sind über SPECIALEFFEKT einsetzbar.
- Mit PICTURE und PICTUREPOSITION kann eine Bitmap eingesetzt werden.

✘ Die Einstellung der Eigenschaften eines Bezeichnungsfeldes ist zwar auch zur Laufzeit des Programms möglich, es empfiehlt sich aber es nur dann im Programm zu tun, wenn Inhalte des Feldes dynamisch zur Laufzeit verändert werden sollen. Ansonsten empfiehlt es sich, die Eigenschaften im Dialogfenster festzulegen.

12.5.4 Schaltflächen, Wechselschaltflächen (CommandButton, ToggleButton)

Die Schaltflächen – Definition ist recht einfach. Sowohl die normalen Schaltflächen als auch die Umschaltbuttons (Wechselschaltflächen) besitzen gleiche Definitions – Eigenschaften.

Der Unterschied besteht darin, daß die ToggleButtons nach dem Aktivieren (anklicken oder Enter – Taste betätigen) solange im gedrückten Zustand verbleiben, bis sie wieder angeklickt werden.

Abbildung 33: Unterschiedliche Formen des Schaltflächen

Wichtigste Eigenschaften:

- Der darzustellende Text wird über die Eigenschaft CAPTION definiert.
- Mit PICTURE kann eine Bitmap – Grafik statt Beschriftung eingefügt werden.
- Im Falle einer eingefügten Grafik kann mit CONTROLTIPTEXT ein gelber Infotext definiert werden.
- Mit PICTUREPOSITION wird die Position der Grafik festgelegt.
- Soll die Schaltflächengröße an den Inhalt angepaßt werden, geschieht es über AUTOSIZE.
- In Tabellenblättern sollte TAKEFOCUSONCLICK auf FALSE gesetzt werden, um zu verhindern, daß die Schaltfläche beim Anklicken den Eingabefocus erhält.
- Der aktuelle Zustand kann über VALUE abgefragt werden.

12.5.5 Textfelder (TextBox)

Textfelder ermöglichen die Eingabe von Texten. Die Eigenschaften dieser Felder sind größtenteils mit den der Bezeichnungsfelder (Label) identisch, weswegen an dieser Stelle auf die nochmalige Beschreibung identischer Eigenschaften verzichtet wird.

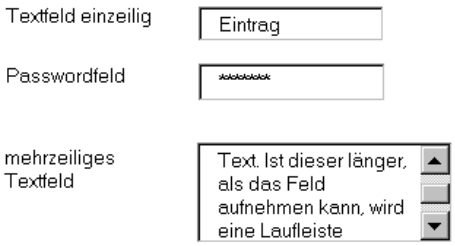


Abbildung 34: Formen von Textfeldern

Wichtigste zusätzliche Eigenschaften:

- Mehrzeilige Textfelder werden mit MULTILINE definiert.
- Bei mehrzeiligen Textfeldern können mit SCROLLBARS Laufleisten eingeblendet werden, wenn der Text über die definerte Breite hinausgeht.
- ENTERKEYBEHAVIOR = TRUE bewirkt einen Zeilenumbruch bei Betätigung der ENTER – Taste (MULTILINE muß auf TRUE gesetzt sein).

- Mit ENTERFIELDBEHAVIOR = 0 wird beim Aktivieren des Textfeldes der gesamte Inhalt markiert, was die Neueingaben, insbesondere bei einzeiligen Feldern praktischer macht.
- WORDWRAP bewirkt einen Zeilenumbruch, wenn der Text den rechten Rand erreicht (erfordert MULTILINE = TRUE).
- Die Textausrichtung wird mit TEXTALIGN definiert.
- Der Zugriff auf den Inhalt erfolgt über TEXT.
- Die Anzahl Zeilen kann mit LINECOUNT, die aktuelle Zeile mit CURLINE die Anzahl Zeichen durch LEN(FELDNAME.TEXT) ermittelt werden.
- Soll das Textfeld als Passwordfeld (Passwordeingabe) dienen, können mit PASSWORDCHAR Zeichen definiert werden, die statt des eingegebenen Textes erscheinen.
- Die Auswahl SELECTMARGIN = TRUE erzeugt einen Leerraum am linken Rand, was eine bequeme Markierung von zeilen in mehrzeiligen Textfeldern möglich macht.

☒ Auf den markierten Text kann über SELTEXT zugegriffen werden. Die Eigenschaften SELSTART und SELLENGHT geben die Position des ersten markierten Zeichens und die Länge der Markierung zurück. Damit kann per Programm Markiert oder eine Textmarkierung manipuliert werden:

```
With Textfeld
  .SelLength = 0        'Löschen der Markierung
  .SelStart = 0         '
  .SelLength = 8        'die ersten 8 Zeichen markieren
  .SelText = ""         'markierter Text wird gelöscht
  .SelStart = 15        'Cursor auf neue Position
  .SelText = "ein"      'Zeichenkette an neuer Position einfügen
End With
```

☒ Die Methoden CUT bzw. COPY übertragen den markierten Text in die Zwischenablage (Ausschneiden, Kopieren), die Methode PASTE überträgt den Inhalt der Zwischenablage in den markierten Bereich des Textes.

12.5.6 Listen, Kombinationsfelder (ListBox, ComboBox)

Listenfelder erlauben Auswahlen von Alternativen ohne direkte Eingabe, Kombinationsfelder bieten neben der gleichen Technik zusätzlich noch als Kombination von Listen- und Textfeldeigenschaften Eingabemöglichkeiten an.

Abbildung 35: Einfache Listenfelder

Wichtigste Eigenschaften:

- Der Zellbereich, aus dem die Werte für die Liste entnommen werden sollen wird über LISTFILLRANGE definiert (in der obigen Abbildung A2:A7).
- Die Zelle, in die der markierte Eintrag ausgegeben werden soll, wird über LINKEDCELL zugeordnet (in der obigen Abbildung die Zellen E2 und E6).
- Eine Titelzeile im Feld kann über COLUMNHEADS = TRUE erzeugt werden. Als Titeleintrag wird der Inhalt der Zelle genommen, die unmittelbar über dem LISTFILLRANGE – Bereich liegt (in der obigen Abbildung A1 – siehe oberes Listenfeld).
- Die Darstellungsform wird über LISTSTYLE definiert.

Voreingestellt ist LISTSTYLE = FMLISTSTYLEPLAIN (unteres Listenfeld in obiger Abbildung), mit FMLISTSTYLEOPTION werden in die Liste runde Optionsflächen eingeblendet (oberes Listenfeld).

- Der erste in der Liste erscheinende Eintrag aus dem LISTFILLRANGE – Bereich wird über TOPINDEX definiert. Voreingestellt ist für TOPINDEX der Wert 0 (= erster Eintrag).

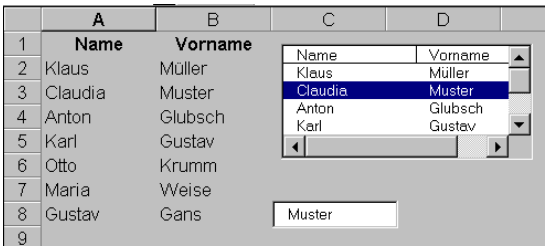


Abbildung 36: Mehrspaltiges Listenfeld

- Mehrspaltige Listenfelder werden erzeugt, wenn der Wert von COLUMNCOUNT größer als 1 ist (obige Abbildung: =2).
- Über BOUNDCOLUMN kann bestimmt werden, welche Spalte eines mehrspaltigen Listenfeldes als Ergebniswert in der Eigenschaft VALUE oder in der verknüpften Zelle (LINKEDCELL) angezeigt werden soll.
- Soll eine Mehrfachauswahl aus dem Feld möglich sein, muß die Eigenschaft MULTISELECT auf den Wert FMMULTISELECTMULTI (ausgewählt wird jede markierte Zeile) oder FMMULTISELECTEXTENDED (die in Windows übliche Mehrfachauswahl mit gedrückter SHIFT- oder STRG – Taste) gesetzt werden.

✘ Die Ausgabe eines Listenfeldes muß nicht in eine Zelle geleitet werden (LINKEDCELL), sie kann in andere Steuerelemente umgeleitet werden. In der wurde die Ausgabe in ein Textfeld geleitet. Beiden Elementen (Listenfeld und Textfeld) wurde bei LINKEDCELL der gleiche Wert zugeordnet – die Zelle C8, über der das Textfeld liegt

Kombinationsfelder sind komplexer als Listenfelder. Sie verfügen über zusätzliche Eigenschaften.

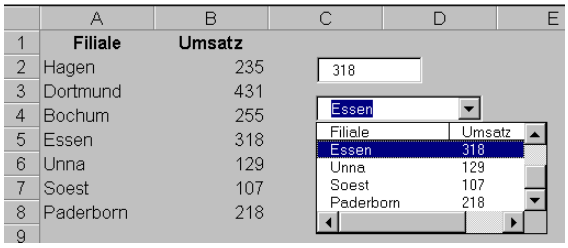


Abbildung 37: Ein mehrspaltiges Kombinationsfeld

Wichtigste zusätzliche Eigenschaften:

- Die Anzahl in der geöffneten Liste angezeigten Zeilen aus dem LISTFILLRANGE – Bereich wird über LISTROWS festgelegt. Enthält dieser Bereich mehr Zeilen, werden Laufleisten eingeblendet.
- Die Breite der geöffneten Liste wird über LISTWIDTH definiert (Angaben in Poinst; 1 Point = 1/72 Zoll). Die Breite des geschlossenen Feldes bleibt davon unberührt.
- SHOWDROPBUTTONWHEN bestimmt, wann der Dropdown - Schalter eingeblendet werden soll (NEVER, ALWAYS, FOCUS).
- STYLE definiert, ob im Feld Eingaben gestattet werden sollen oder ob das Feld nur als unveränderbare Liste zur Verfügung stehen soll.

12.5.7 Drehfelder, Laufleisten (SpinButton,ScrollBar)

Bildlaufleisten und Drehfelder dienen dazu eine ganze Zahl aus einem vordefinierten Wertebereich zu selektieren. Der zulässige Zahlenbereich liegt im LONG – Zahlenraum (etwa + / - 2.10^9).

Bei Laufleisten kann der Wert durch das Anklicken einer der Inkrement- oder Dekrement – Schaltflächen, das Verschieben eines Schiebers oder einen Klick in die Lauffläche verändert werden.

Das Drehfeld ist eine „abgemagerte" Variante des Laufleiste – es enthält nur eine Inkrement- und eine Dekrement – Schaltfläche.

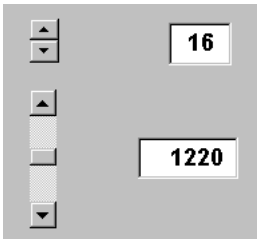


Abbildung 38: Drehfeld
und Laufleiste

Wichtigste Eigenschaften:

- DELAY bestimmt die Verzögerung zwischen Klick und Ergebnis in Milisekunden.
- LINKEDCELL bildet die Verbindung zur Ausgabezelle eines Tabellenblattes. Ausgabe muß nicht in eine Tabellenzelle erfolgen (s. oben).
- Bei der Laufleiste kann über LARGECHANGE die seitenweise Wertänderung (Klick in die Lauffläche) definiert werden.
- Die Grenzen des zulässigen Wertebereichs werden mit MIN / MAX festgelegt werden.
- ORIENTATION bestimmt die Ausrichtung des Steuerelements (vertikal / horizontal).
- Die Abmessungen des Schiebers einer Laufleiste werden mit PROPORTIONALTHUMP festgelegt. Steht der Wert auf TRUE, so ist die Schiebergröße umgekehrt proportional zur Größe des Wertebereichs. Bei großen Wertebereichen auf FALSE setzen, sonst ist der Schieber kaum sichtbar und bedienbar !
- SMALLCHANGE legt die Schrittweite der Wertänderung beim Klick auf die Inkrement- bzw. Dekrement – Schaltflächen fest.
- Erzeugte Werte können über VALUE abgefragt werden.

12.5.8 Kontrollkästchen, Optionsfelder (CheckBox, OptionButton)

Kontrollkästchen (in der unteren Abbildung die rechte Reihe von Elementen) eignen sich in für Ja / Nein – Entscheidungen. Der aktuelle Zustand wird durch ein Häkchen ✔ im quadratischen Fenster des Kontrollkästchens angezeigt.

Abbildung 39: Options- und Kontrollfelder

Optionsfelder (linke Reihe in der Abbildung) unterscheiden sich otisch aber auch inhaltlich von Kontrollkästchen – Optionsfelder können nur einzeln aktiviert werden (Punkt in dem kreiförmigen Fenster), die Aktivierung eines Optionsfeldes deaktiviert ein eventuell davor schon aktives.

Wichtigste Eigenschaften:

- Der aktuelle Zustand kann über die Eigenschaft VALUE erfragt werden. Zulässige Werte sind FALSE, TRUE und NULL.
- Die Beschriftung kann über CAPTION festgelegt werden.
- Über die Eigenschaft GROUPNAME können Optionsfelder und Kontrollfelder gruppiert werden. Sollen mehrere Optionsfelder zu einer Gruppe gehören, wird bei allen unter GROUPNAME der gleiche Gruppenname angegeben.

✘ Zu einer Gruppe gehörende Felder müssen nicht optisch zusammengefasst sein. Sie können über das ganze Tabellenblatt verteilt sein.

✘ Options- und Kontrollfelder besitzen ebenfalls die Eigenschaft LINKEDCELL. Gruppierten Felder dürfen in dieser Eigenschaft nicht die gleichen Zellen zugeordnet werden, da eine Zelle keine zwei gegensätzlichen Werte gleichzeitig aufnehmen kann.

12.5.9 Verbindung Zelle – Steuerelement

Die Verbindung (der Datenaustausch) zwischen Zellen und Steueerelementen wurde schon mehrfach im Zusammenhang mit Eigenschaften von Steuerelementen erwähnt. Hier nochmal ein deutlicher Hinweis:

Die Verknüpfung zwischen Zelle und Steuerelement ist über die Eigenschaft VALUE oder TEXT der Steuerelemente möglich. Eingaben im Steuerelement werden über diese Eigenschaften in Zellen übertragen – aber auch umgekehrt: ein Eintrag in die „angelinkte“ Zelle bewirkt eine Änderung des Wertes im Steuerelement.

Die Verbindung wird über die Eigenschaft LINKEDCELL hergestellt.

✘ Eine Ausnahme bilden die Steuerelemente *Schalter* und *Bezeichnungsfeld*, die keine VALUE – Eigenschaft besitzen.

12.5.10 Blattschutz

Tabellenblätter, die als Steuerformulare benutzt werden, sind weiterhin normale Tabellen, in die insbesondere Werte eingetragen werden können, auch wenn vielleicht das Gitternetz (evtl. auch die Laufleisten) ausgeblendet wurde und damit optisch nicht der Eindruck eines Tabellenblattes entsteht. Um Eingaben zu verhindern, sollte der Blattschutz aktiviert werden (Funktionskombination EXTRAS / SCHUTZ / BLATTSCHUTZ). Dabei ist darauf zu achten, daß die Option OBJEKTE des Blattschutz – Dialogs aktiv ist.

Nach der Aktivierung des Blattschutzes sind die Steuerelemente weiter benutzbar, manuelle Eingaben in das Blatt sind allerdings nicht mehr möglich.

✘ **Achtung**: Wird das Tabellenblatt gleichzeitig als Steuerformular und als Datenformular benutzt (siehe Beispielprogramm 1 weiter oben im Text), ist nach der Aktivierung des Blattschutzes keine Dateneingabe oder Korrektur im Tabellenblatt möglich ! Lösung des Problems – siehe *Beispielprogramm 2*.

12.6 Beispielanwendung 2

Die in der Beispielanwendung 1 vorgestellte Lösung soll nun mit Hilfe der Steuerelemente aus der Steuerelemente – Toolbox realisiert werden. Dazu wird das als Formular benutzte Tabellenblatt umgestaltet – der Datenbereich wird daraus entfernt und in ein eigenes Blatt versetzt, damit das Formular für Eingaben gesperrt werden kann. Gleichzeitig werden eineige recht umfangreiche Veränderungen am Programm vorgenommen, um die oben im Text beschriebenen Eigenschaften der benutzten Steuerelemente zu demonstrieren.

Das Formularblatt wurde „vorbehandelt“ (ohne Gitternetz, Laufleisten, Symbolleisten usw.), wie schon weiter oben beschrieben und mit den entsprechenden Elementen versehen:

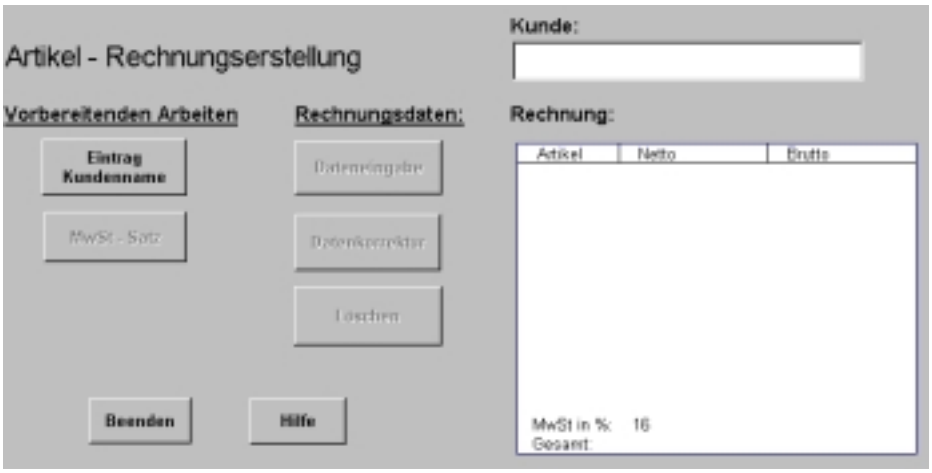


Abbildung 40: Formularblatt Programmbeispiel 2

Die Bedienung des Formulars ist über Schaltflächen möglich. Nach dem Aufruf bleiben allerdings alle Schaltflächen inaktiv, bis auf eine – *Eintrag Kundenname*. Sie übrigen werden erst dann aktiviert, wenn der in der Reihenfolge der Bearbeitung nächste Schritt erfolgen soll.

Das Formularblatt ist für direkte Eingaben gesperrt.

Die vorbereitenden Aktionen sind in der AUTO_OPEN - Prozedur der Arbeitsmappe zusammengefaßt:

```
Sub Auto_open()
  Worksheets("Hilfe").Visible = False
  Worksheets("Rdaten").Visible = False
  Sheets("Eingangsblatt").Select
  Anfang.cmdEingabe.Enabled = False
  Anfang.cmdMwSt.Enabled = False
  Anfang.cmdKorrektur.Enabled = False
  Anfang.cmdLöschen.Enabled = False
  Anfang.txtKunde.Text = ""
  ActiveSheet.Protect DrawingObjects:=True, _
                      Contents:=True
End Sub
```

Die einzige aktive Schaltfläche ist die Schaltfläche Eintrag Kundenname , die Angabe eines Kundennamens und den Eintrag ins Textfeld Kunde möglich macht. Der Kundenname wird über eine InputBox eingetragen:

```
Private Sub cmdKunde_Click()
  Anfang.txtKunde.Text = InputBox("Bitte Kundennamen eingeben:", "Kunde")
  Anfang.cmdMwSt.Enabled = True
End Sub
```

Gleichzeitig aktiviert die Schaltfläche das nächste Element in der Bearbeitungsreihenfolge – die Schaltfläche *MwSt-Satz*.

Diese ermöglicht wiederum über eine INPUTBOX die Eingabe des für die Berechnung geltenden MwSt – Satzes

```
Private Sub cmdMwSt_Click()
  Worksheets("RDaten").Range("B15").Value = _
  Val(InputBox("Bitte MwSt-Satz eingeben:", "MwSt-Satz"))
  Anfang.cmdEingabe.Enabled = True
End Sub
```

und aktiviert die nächste Schaltfläche – *Dateneingabe*.

Diese wiederum bereitet das Datenblatt der Rechnung (*Rdaten*) vor, trägt darin Formate und Formeln für die Berechnungen ein:

```
Private Sub cmdEingabe_Click()
 Application.ScreenUpdating = False
  Worksheets("RDaten").Activate
   ActiveWindow.DisplayZeros = False
   Worksheets("RDaten").Range("B2:B13").Style = "Currency"

   With Worksheets("RDaten").Range("C2:C12")
      .Formula = "=B2+B2*$B$15/100"
      .Style = "Currency"
   End With
    Worksheets("RDaten").Range("A2").Select
    Application.DisplayAlerts = False
     ActiveSheet.ShowDataForm
    Application.DisplayAlerts = True
    Worksheets("RDaten").Range("C15").Formula = "=sum(C2:C12)*$B$15/100"
    Worksheets("RDaten").Range("C16").Formula = "=sum(C2:C12)"
    Worksheets("RDaten").Range("C2:C12").Select
    For Each Zelle In Selection
       If Zelle.Value = 0 Then
         Zelle.Value = ""
       End If
    Next
  Application.ScreenUpdating = True
  Worksheets("Eingangsblatt").Activate
  Anfang.cmdKorrektur.Enabled = True
  Anfang.cmdLöschen.Enabled = True

End Sub
```

Nach der Aufbereitung des Tabellenblattes wird die interne EXCEL – Datenmaske für die Datenerfassung aufgerufen. Eventuelle System – Fehlermeldungen werden unterdrückt (DISPLAYALERTS = FALSE / TRUE).

Damit der Wechsel des Arbeitsblattes und die Arbeitsvorgänge im Blatt nicht sichtbar werden, wird über SCREENUPDATING = FALSE / TRUE die Bildschirmaktualisierung abgeschaltet und später wieder aktiviert.

Die Aktivierung des Eingangsblates (incl. Schaltflächen für Datenkorrektur und Löschen) werden die Daten des Datenblattes in das Textfeld rechts im Eingangsblatt übernommen (das Feld ist beim Aufbau des Formularblattes über die Eigenschaft LISTFILLRANGE mit dem entsprechenden Bereich der Tabelle verknüpft).

Für die Datenkorrektur wird ein eigenes Blatt angelegt und *Korrektur* genannt. In dieses Blatt wird der Inhalt des Datenblattes übertragen, das Hilfsgitter sowie Zeilen- und Spaltennumerierung werden ausgeblendet. Die nicht ausgefüllten Bereiche sowie Formelzellen werden ermittelt und gelöscht.

Für die Datenkorrektur wird wieder die interne Datenmaske eingeblendet.

Nach der Korrektur werden die Daten in das Datenblatt übertragen und anschließend das nicht mehr benötigte Korrekturblatt gelöscht und das Eingangsblatt aktiviert.

Das Datenblatt wird „versteckt“.

Der Aufruf des Eingangsblattes zieht die automatische Aktualisierung der im Textfeld dieses Blattes angezeigten Rechnungsdaten nach sich.

Für die Abwicklung der Datenkorrektur sind die folgenden Anweisungen vorgesehen:

```
Private Sub cmdKorrektur_Click()
    Dim A
    Dim B
    Sheets.Add
    ActiveSheet.Name = "Korrektur"
    Worksheets("Rdaten").Visible = True
    Sheets("Rdaten").Select
    ActiveSheet.Range("A1:B12").Select
    Selection.Copy
    Sheets("Korrektur").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    ActiveWindow.DisplayGridlines = False
    ActiveWindow.DisplayHeadings = False
    With ActiveSheet
      .Range("A1").Select
      .Cells(Rows.Count, ActiveCell.Column).Select
        If IsEmpty(ActiveCell) Then
          ActiveCell.End(xlUp).Select
          A = ActiveCell.Row
          B = ActiveCell.Column
        End If
    End With

    With Worksheets(1)
      .Range(.Cells(A + 1, B), .Cells(A + 9, B + 1)).ClearFormats
      .Range("A2").Select
      .ShowDataForm
      .Range(.Cells(A, B), .Cells(1, B + 1)).Select
    End With
        Application.CutCopyMode = False
        Selection.Copy

    Sheets("Rdaten").Select
    ActiveSheet.Range("A1").Select
    ActiveSheet.Paste

    Sheets("Korrektur").Select
    Application.DisplayAlerts = False
    ActiveWindow.SelectedSheets.Delete
    Application.DisplayAlerts = True
    Worksheets("Rdaten").Visible = False

End Sub
```

Das Löschen wird durch das Einblenden einer Abfrage (MsgBox) eingeleitet:

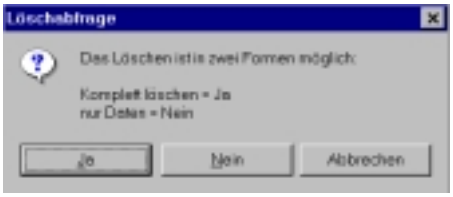


Abbildung 41: Löschabfrage zu Bsp. 2

Je nach betätigter Schaltfläche werden entweder alle Daten gelöscht (incl. Kundenname im Textfeld), nur der Inhalt des Datenblattes (und damit auch der Inhalt des Textfeldes) oder der Vorgang wird abgebrochen. In allen drei Fällen erfolgt die Rückkehr zum Eingangsblatt.

Die Löschprozedur hat den folgenden Inhalt:

```
Private Sub cmdLöschen_Click()
 Antwort = MsgBox("Das Löschen ist in zwei Formen möglich:" & _
      Chr(13) & Chr(13) & "Komplett löschen = Ja" & Chr(13) & _
      "nur Daten = Nein", 35, "Löschabfrage")

 Select Case Antwort

 Case vbYes
 Application.ScreenUpdating = False
  Worksheets("Rdaten").Visible = True
  Worksheets("Rdaten").Select
  Worksheets("Rdaten").Range("A2:C12, C15:C16").ClearContents
  Worksheets("Rdaten").Range("A2").Select
  Worksheets("Rdaten").Visible = False
 Application.ScreenUpdating = True
   Worksheets("Eingangsblatt").Activate
   Anfang.txtKunde.Text = ""
   Anfang.cmdEingabe.Enabled = False
   Anfang.cmdMwSt.Enabled = False
   Anfang.cmdKorrektur.Enabled = False
   Anfang.cmdLöschen.Enabled = False

 Case vbNo
 Application.ScreenUpdating = False
   Worksheets("Rdaten").Visible = True
   Worksheets("Rdaten").Select
   Worksheets("Rdaten").Range("A2:C12, C15:C16").ClearContents
   Worksheets("Rdaten").Range("A2").Select
   Worksheets("Rdaten").Visible = False
 Application.ScreenUpdating = True
   Worksheets("Eingangsblatt").Activate

 Case Else
   Worksheets("Eingangsblatt").Activate
 End Select

  Worksheets("Eingangsblatt").Activate
End Sub
```

Für die Online – Hilfe wurde eine USERFORM benutzt (als „Überleitung“ zum nächsten Kapitel). Sie wird eingeblendet über die der Hilfe – Schaltfläche zugeordnete Prozedur:

```
Private Sub Hilfe_Click()
frmHelp.Show
End Sub
```

Die UserForm hat das folgende Aussehen:

Abbildung 42: Online-Hilfe in einer USERFORM

Sie besteht im wesentlichen aus einem Kombinations- und einem Textfeld, die miteinander verknüpft sind. Als Suchbegriff- und Hilfetextquelle dient eine Tabelle, die in einer Spalte die Suchbegriffe und in einer zweiten die Hilfetexte enthält, jedoch nie sichtbar ist (siehe AUTO_OPEN – Prozedur weiter oben im Text).

Das Kombinationsfeld wurde bei der Generierung über die Eigenschaft ROWSOURCE mit den die Hilfebegriffe enthaltenden Zellen der Tabelle *Hilfe* verknüpft.

Das Textfeld wird über

```
Private Sub cbxWort_Change()
 frmHelp.txtMeld.Text = Hilfe.Cells(1 + frmHelp.cbxWort.ListIndex, 2)
End Sub
```

mit den Hilfetexten aus der gleichen Tabelle versorgt, wobei die Position des Hilfetextes in der Tabelle mit Hilfe des LISTINDEX – Wertes des Kombinationsfeldes (CBXWORT) ermittelt wird.

Das Formular hat nach einem Programmdurchlauf zur Datenerfassung das folgende Aussehen:

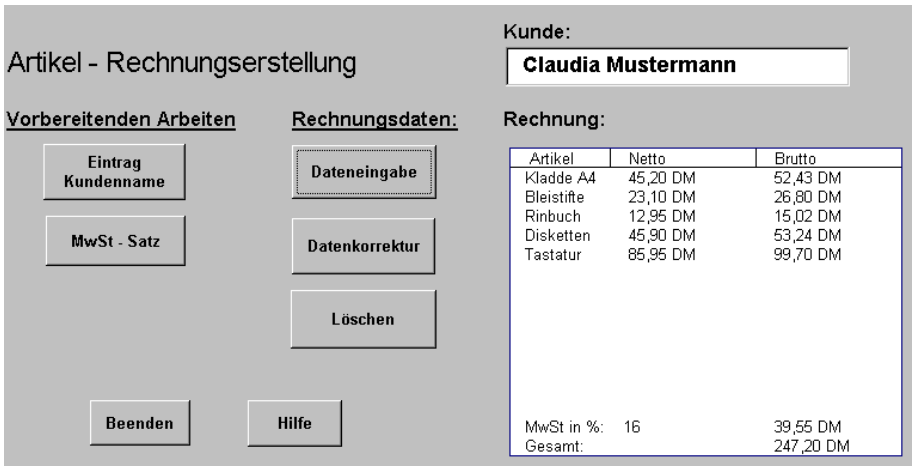


Abbildung 43: Formular aus Programm 2 nach Datenerfassung

➡ Das komplette Programm ist im Anhang zu finden.

➡ Die Form der im Programm benutzen Anweisungen und Anweisungsketten erhebt keinen Anspruch auf „Vollkommenheit“. Auch hier, wie schon im vorherigen Programmbeispiel, existieren elegantere Lösungen. Manches Problem wurde auch hier umständlich und eher umschreibend, dafür aber hoffentlich verständlich gelöst. Das Programm soll allerdings die Anwendung vorgestellter Techniken und Elemente am einfachsten demonstrieren – die Schönheit des Programmcodes behält sich der Autor für einen weiterführenden Kurs.

12.7 Selbstdefinierte Dialoge - UserForm

Die Nachteile eines im Tabellenblatt untergebrachten Formulars, insbesondere seine Veränderbarkeit durch Benutzer, lassen sich durch die Verwendung von Dialogen ausschließen. Dialoge erlauben keinen Zugriff auf ein Tabellenblatt, solange sie aktiv sind.

An dieser Stelle soll nur in kurzer Form auf die Entwicklung eines benutzerdefinierten Dialogs eingegangen werden. Für weitergehende Informationen zum Aufbau und Definition sei auf die Broschüre URZ/B009 verwiesen.

Für den Aufbau eines Userdialogs wird in der VBA – Entwicklungsumgebung über die Funktionskombination EINFÜGEN / USERFORM ein leeres Dialogformular erzeugt. Gleichzeitig wird die für dem Formularaufbau benötigte Werzeugsammlung eingeblendet:

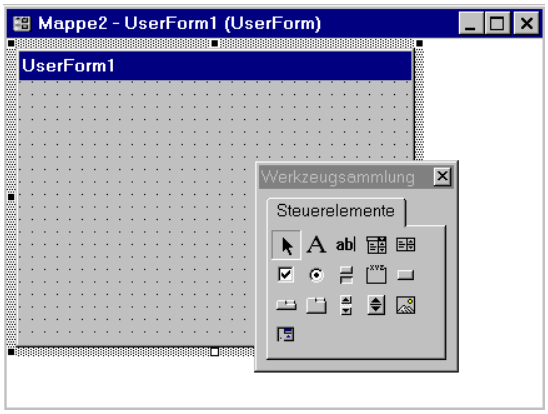


Abbildung 44: VBA - Dialogfenster
mit Werkzeugsammlung

Der Dialog wird ähnlich wie die bisher beschriebenen, in Tabellenblättern untergebrachten, Formulare aufgebaut (Steuerelement aus der Werkzeugsammlung in Formular ziehen, Eigenschaften über das Eigenschaftsfenster definieren).

12.7.1 Beispiel 1

Das nachfolgende kleine Beispielformular wurde aus einem Textfeld und zwei Schaltflächen aufgebaut. Es soll die Aufgabe erfüllen, den eingegebenen Wert eines MwSt – Satzes in die Zelle eines Tabellenblattes einzufügen:

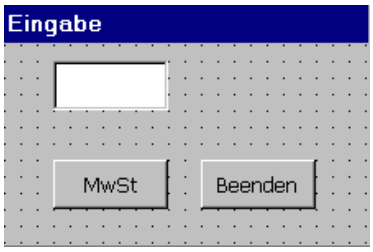


Abbildung 45: Einfacher VBA-Dialog

Die Funktionalität wird durch Prozeduren (wie bei bisherigen Anwendungen) gewährleistet. Es Ergibt sich jedoch ein wesentlicher Unterschied – ein Formular im Tabellenblatt ist mit dem Tabellenblatt immer sichtbar (es sei denn, das Tabellenblatt wird ausgeblendet), eine UserForm muß explizit eingeblendet werden. Dies geschieht durch die Anweisung:

```
Private Sub Anfang()
  Eingabe1.Show
End Sub
```

(hier in einer eigenen Sub untergebracht).

Die obige UserForm ist so konzipiert, das sobald nach dem Einblenden die Maus innerhalb der Form bewegt wird, das Eingabefeld den Focus erhält (aktiviert wird), dami die Eingabe möglich wird, ohne daß dieses Feld angeklickt werden muß.

Die Aktivierung des Texfeldes (Name = *Wert1*) erfolgt die MOUSEMOVE – Prozedur des Formulars:

```
Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift
As Integer, ByVal X As Single, ByVal Y As Single)
Wert1.SetFocus
End Sub
```

Nach der Eingabe des Wertes wird über die Schaltfläche *MwSt* (Name = *Steuer*) in eine Zelle der *Tabelle2* eingetragen:

```
Private Sub Steuer_Click()
Worksheets("Tabelle2").Range("A3").Value = Eingabe1.Wert1.Value
End Sub
```

Die Schaltfläche *Ende* beendet die Anwendung und schließt die Form:

```
Private Sub Ende_Click()
 End
End Sub
```

12.7.2 Beispiel 2

Das Formular des folgenden Beispiels soll Einträge für Artikel und Preise in zwei Zellen eines Tabellenblattes vornehmen. Es soll eingeblendet werden, sobald das Tabellenblatt aktiviert wird. Dazu wird die Aktivierung in der WORKSHEET_ACTIVATE - Prozedur des Tabellenblattes mit SHOW aktiviert und gleichzeitig im Blatt die Zelle A2 ausgewählt, damit der erste Eintrag in dieser Zelle erfolgen kann[6]:

```
Private Sub Worksheet_Activate()
 Range("A2").Activate
 frmArtikel.Show
End Sub
```

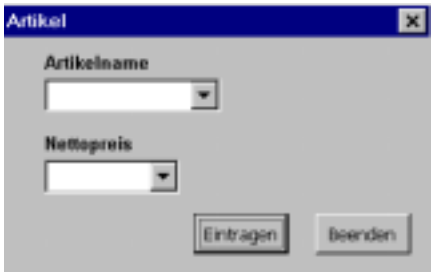


Abbildung 46: Einfacher VBA-Dialog (2)

Artikel und Preise können im Dialogfeld aus ComboBoxen entnommen werden. Diese werden in der Initialisierungs – Prozedur der Form mit Werten belegt[7]:

```
Private Sub UserForm_Initialize()
With cmbArtikel
  .AddItem "Heft"
  .AddItem "Hefter"
  .AddItem "Bleistift"
  .AddItem "Filzschreiber"
  .AddItem "Ordner"
  .AddItem "Tinte"
End With
With cmbPreis
  .AddItem "1.25"
  .AddItem "2.95"
  .AddItem "1.15"
  .AddItem "1.98"
  .AddItem "3.75"
  .AddItem "2.30"
End With
End Sub
```

✘ Die ComboBoxen sind so definiert, daß Werte in deren Eingabefelder auch manuell eingetragen werden können.

[6] Soll der Eintrag in der ersten freien Zelle der Spalte A erfolgen, muß diese vorher ermittelt werden. Siehe dazu Beispiel zur Ermittlung einer freien Zelle weiter oben im Text.

[7] Die Werte können auch Tabellenzellen entnommen werden. Siehe dazu ROWSOURCE – Eigenschaft der COMBOBOX. In diesem Fall ist eine Initialisierung in der obigen Form nicht nötig.

Über die Schaltfläche *Eintragen* werden die Werte ins Tabellenblatt eingetragen:

```
Private Sub cmdEintrag_Click()
 If cmbArtikel.Text = "" Or cmbPreis = "" Then
  MsgBox "Artikel und Preis eintragen !"
 Else
  Cells(ActiveCell.Row, 1).Value = cmbArtikel.Text
  With Cells(ActiveCell.Row, 2)
   .Value = cmbPreis.Value
   .Style = "Currency"
  End With
  Cells(ActiveCell.Row + 1, 1).Activate
 End If
End Sub
```

Vor dem Eintrag wird geprüft, ob beide Felder Einträge enthalten, im Negativfall wird eine Meldung ausgegeben und kein Eintrag vorgenommen.

Beim Eintrag wird der Zelle mit dem Preis das Währungsformat zugewiesen. Anschließend wird die Zelle der Spalte 1 (A) eine Zeile tiefer aktiviert, damit ein neuer Eintrag erfolgen kann.

✘ Die Initialisierungswerte und manuell eingegebene Werte des Preisfeldes sollten Dezimalpunkte statt Kommata enthalten. Beim Übertragen in die Zelle werden diese in ein Komma umgewandelt (richtige EXCEL – Notation).

✘ Solange die Form aktiv bleibt, ist keine direkte manuelle Eingabe in die Zellen des Tabellenblattes möglich.

✘ Soll das Formular sofort nach dem Öffnen der Arbeitsmappe in die aktivierte Tabelle eingeblendet werden, sollte in der Workbook_Open – Prozedur für die entsprechende Aktivierung des Tabellenblattes, der richtigen Zelle und das Einblenden des Formulars gesorgt werden:

```
Private Sub Workbook_Open()
 Worksheets(1).Activate
 Range("A2").Activate
 frmArtikel.Show
End Sub
```

12.7.3 Beispiel 3

Das nachfolgende Beispiel für die Anwendung von USERFORMS ist etwas umfangreicher, aber dennoch einfach in der Programmstruktur. Auch hier wird wiederum kein Wert auf komplizierte profesionelle Programmierung gelegt; die Anweisungen sollen einfach und verständlich sein.

Das zugrundeliegende Programm soll die folgenden Aufgaben erfüllen:

- Über eine USERFORM mit dem folgenden Design:

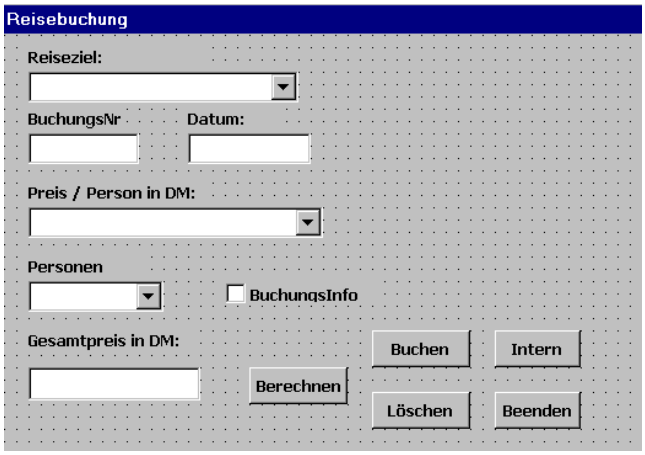


Abbildung 47: UserForm des Beispiels 3

sollen in ein Tabellenblatt Buchungssätze eingetragen werden. Dabei sind folgende Bedingungen zu erfüllen:

- Die Reiseziele, Preise pro Person und Reiseziel sowie Personenzahl sollen aus ComboBoxen entnommen werden können. Diese sollen nach dem Laden des Formulars gefüllt zur Verfügung stehen. Die Daten stehen in einem eigenen Tabellenblatt mit dem Namen *Daten*.
- Die Buchungsnummer soll um 1 höher sein als die höchste im Tabellenblatt eingetragene. Auch sie soll nach dem Laden des Formulars schon im Feld erscheinen.
- Das Feld *Datum* soll nach dem Laden das aktuelle Tagesdatum enthalten.
- Die übrigen Felder sollen nach dem Laden leer sein.
- Die Schaltflächen *Buchen* und *Löschen* sollen nach dem Laden der Form nicht sichtbar sein, sie sollen erst dann erscheinen, wenn der Gesamtpreis über die Schaltfläche *Berechnen* errechnet wurde.
- Für eine Buchung kann ein fester Rabatt vergeben werden, wenn die CHECKBOX *Buchungsinfo* aktiv ist. Der Betrag soll ausgerechnet werden, im Tabellenblatt soll ein Hinweis auf Rabatt eingetragen werden.
- Die Schaltfläche *Intern* soll die Gesamtsumme aller Gesamtpreise im Tabellenblatt in einer eigenen UserForm anzeigen:

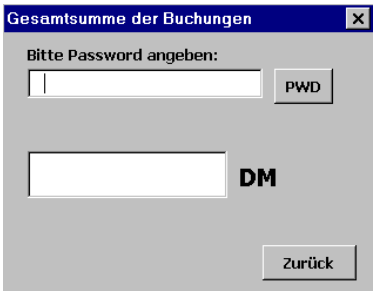


Abbildung 48: UserForm mit Password - Abfrage

allerdings geschützt durch ein Password.

- Die Funktionen der übrigen Schaltflächen – siehe weiter unten im Text.

Die Initialisierungsprozedur der Form:

```
Private Sub UserForm_Initialize()
cmbZiel.RowSource = "Daten!A1:A12"
With cmbPreis
  .ColumnCount = 2
  .RowSource = "Daten!A1:B12"
  .TextColumn = 2
End With
cmbPers.RowSource = "Daten!D1:D10"
  txtDatum.Text = Date
  cmdBuch.Visible = False
  cmdLösch.Visible = False

ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select

If ActiveCell.Row = 1 Then
  txtNr.Text = "0001"
Else
  txtNr.Text = Cells(ActiveCell.Row, 1).Value + 1
End If
  Cells(ActiveCell.Row + 1, 1).Activate
  Selection.NumberFormat = "0000"
End Sub
```

füllt die COMBOBOXES mit Werten aus der Tabelle DATEN.

Die Box für den Preis wird mit .COLUMNCOUNT zweispalig definiert wird. Mit .TEXTCOLUMN wird festgelegt aus welcher der Spalten der Wert für die TEXT – Eigenschaft der ComboBox genommen werden soll.

Mit

```
txtDatum.Text = Date
  cmdBuch.Visible = False
  cmdLösch.Visible = False
```

wird das Tagesdatum eingesetzt und die Schaltflächen *Buchen* und *Löschen* „versteckt“.

Die Zeile

```
ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
```

ermittelt über .SpecialCells die letzte belegte Zelle des aktiven Tabellenblattes.

Die nachfolgenden Anweisungen belegen in Abhängigkeit davon, in welcher Zeile die letzte belegte Zelle gefunden wurde das Feld mit der Buchungsnummer entweder mit einer *0001* (leere Tabelle mit Überschriftenzeile vorgefunden) oder mit einem Wert, der um 1 höher ist als die letzte vorgefundene Buchungsnummer.

Abschließend wird die Tabelle für eine neue Buchung vorbereitet – die Zelle unter der letzten Buchungsnummer wird aktiviert und mit einem vierstelligen numerischen Format belegt.

Ist das Formular ausgefüllt, wird über die Schaltfläche *Berechnen* der Gesamtpreis ausgerechnet:

```
Private Sub cmdRechnen_Click()
  If chkBuch.Value = True Then
   txtPreis.Text = Val(cmbPreis.Text) * Val(cmbPers.Value) * 0.95
  Else
   txtPreis.Text = Val(cmbPreis.Text) * Val(cmbPers.Value)
  End If
  cmdBuch.Visible = True
  cmdLösch.Visible = True
End Sub
```

In Abhängigkeit davon, ob Rabatt gewährt werden soll oder nicht, werden die entsprechenden Multiplikationen (*Einzelpreis* * *Personenzahl*) mit 0,95 oder voll berechnet.

Gleichzeitig werden die Schaltflächen *Buchen* und *Löschen* aktiviert.

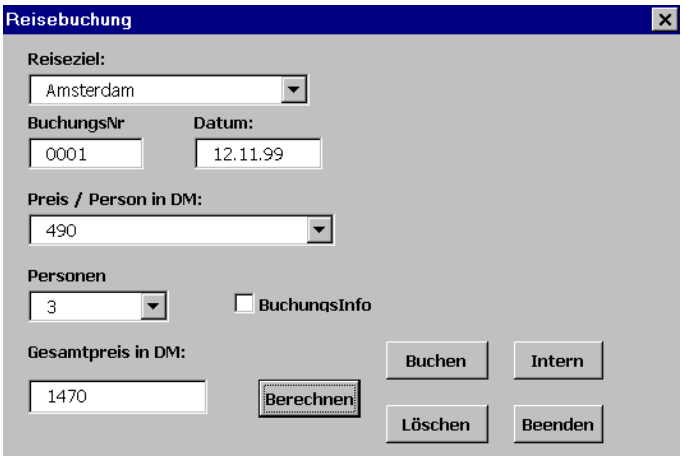


Abbildung 49: Ausgefülltes Buchungsformular

Das Buchen (Eintragen der Daten ins Tabellenblatt) erfolgt über die CLICK – Prozedur der Schaltfläche *Buchen*:

```
Private Sub cmdBuch_Click()
 Dim Zeile As Integer, Spalte As Integer
 Zeile = ActiveCell.Row
 Spalte = ActiveCell.Column
 ActiveCell.Value = txtNr.Text
 Selection.NumberFormat = "0000"
 Cells(Zeile, Spalte + 1).Value = CDate(txtDatum.Text)
 Cells(Zeile, Spalte + 2).Value = cmbZiel.Text
 Cells(Zeile, Spalte + 3).Value = CCur(cmbPreis.Text)
 Cells(Zeile, Spalte + 4).Value = cmbPers.Text
 Select Case chkBuch.Value
   Case True
    Cells(Zeile, Spalte + 5).Value = "Ja"
   Case False
    Cells(Zeile, Spalte + 5).Value = ""
 End Select
 Cells(Zeile, Spalte + 6).Value = CCur(txtPreis.Text)
End Sub
```

Da bei der Initialisierung schon die Zelle für die Buchungsnummer aktiviert wurde, werden deren Spalten- und Zeilenadresse als Grundwerte für die Verteilung der Feldinhalte in der Tabelle genommen. Datums- und Währungsangaben werden über entsprechende Funktionen (CDATE und CCUR) beim Eintrag berücksichtigt.

Die Schaltfläche Löschen setzt das Formular in den Ursprungszustand zurück:

```
Private Sub cmdLösch_Click()
 cmdBuch.Visible = False
 ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
 If ActiveCell.Row = 1 Then
  txtNr.Text = "0001"
 Else
  txtNr.Text = Cells(ActiveCell.Row, 1).Value + 1
 End If
 Cells(ActiveCell.Row + 1, 1).Activate
 cmbZiel.Text = ""
 cmbPreis.Text = ""
 cmbPers.Text = ""
 chkBuch.Value = False
 txtPreis.Text = ""
 cmdBuch.Visible = False
 cmdLösch.Visible = False
End Sub
```

Die Folgebuchungsnummer wird eingesetzt, das Tagesdatum bleibt im Feld.
Die Schaltflächen *Buchen* und *Löschen* werden ausgeblendet.

Die Schaltfläche *Intern* blendet ein zweites Formular ein – das passwordgeschützte Formular für die Gesamtsumme der Reisekosten:

```
Private Sub cmdGesamt_Click()
 Dim Pwd As String
  frmGesamt.Show
End Sub
```

Nach der Eingabe des Passwords wird dieses nach einem Klick auf die Schaltfläche *PWD* geprüft:

```
Private Sub cmdPwd_Click()
 If txtPwd.Text = "Gogolok" Then
   ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
   txtSumme.Value = _
   WorksheetFunction.Sum(Range(ActiveCell, Cells(2, ActiveCell.Column)))
  Else
   MsgBox "Falsche Angabe, bitte wiederholen !", 16, "Fehler"
   txtPwd.Text = ""
   txtPwd.SetFocus
 End If
End Sub
```

Ist das Password richtig (hier der Name des Verfassers), wird die Summe der Beträge im Zellbereich zwischen der letzten belegten Zelle und der Zelle in der Zeile 2 der gleichen Spalte ausgerechnet.

 Für die Berechnung wird mit

```
WorksheetFunction.Sum(.....)
```

die EXCEL – Standardfunktion SUMME benutzt.

Ist das angegebene Password falsch, erscheint eine Fehlermeldung, die Eingabe wird gelöscht und das Passwordfeld für eine neue Eingabe aktiviert.

Die Schaltfläche *Zurück* dieser Form, blendet die Form aus, womit die Anwendung zum Buchungsformular zurückblendet:

```
Private Sub cmdBack_Click()
  Hide
End Sub
```

Die Buchungstabelle erhält im Laufe der Buchungen das folgende Aussehen.

	A	B	C	D	E	F	G
1	**Buchungs#**	**Datum**	**Reiseziel**	**Preis / Person**	**Personen**	**Skonto**	**Gesamtpreis**
2	0001	12.11.99	Hamburg	399,00 DM	3	Ja	1.137,15 DM
3	0002	12.11.99	Kopenhagen	575,00 DM	5		2.875,00 DM
4	0003	12.11.99	München	385,00 DM	5	Ja	1.828,75 DM
5	0004	12.11.99	Hamburg	399,00 DM	3	Ja	1.137,15 DM
6	0005	12.11.99	Berlin	345,00 DM	4		1.380,00 DM
7	0006	12.11.99	Stuttgart	280,00 DM	8	Ja	2.128,00 DM
8							

Abbildung 50: Buchungstabelle des Programmbeispiels

 Das Programm besitzt „indirekte Buchungssicherung“ – wird eine Buchung mit der Taste *Beenden* abgebrochen, entsteht in der Buchungstabelle eine leere Zeile, die Folgebuchung beginnt dann wieder mit der Buchungsnummer 0001. Um die Kontrolle über solche „Fehlbuchungen“ zu behalten, wurde im Programm keine Sonderbehandlung solcher Fälle vorgesehen.

13 Anhang

13.1 Beispielprogramm 1

```
Sub Auto_open()
  Worksheets("Hilfe").Visible = False
  Sheets("Auswertung").Select
  Range("A4").Select
End Sub

'=====================================================
'     Eintrag Kundenname
'=====================================================

Sub Kunde()
    Range("Auswertung!E1") = "Kunde: " & Application.InputBox( _
    prompt:="Geben Sie den Kundennamen ein:", Type:=2)
    Range("A2").Select
End Sub

'=====================================================
'     Eintrag MwSt
'=====================================================

Sub Steuer()
    Range("Auswertung!B15") = Application.InputBox( _
    prompt:="Geben Sie den MwSt-Satz ein:", Type:=1)
    Range("A2").Select
End Sub
'=====================================================
'  Dateneintrag
'=====================================================
Sub Eintrag()

   ActiveWindow.DisplayZeros = False
   Range("B2:B13").Select
   Selection.Style = "Currency"
   Range("C2:C12").Select
    With Selection
      .Formula = "=B2+B2*$B$15/100"
      .Style = "Currency"
    End With
    Range("A2").Select
    Application.DisplayAlerts = False
     ActiveSheet.ShowDataForm
    Application.DisplayAlerts = True
    Range("C15").Formula = "=sum(C2:C13)*$B$15/100"
    Range("C16").Formula = "=sum(C2:C13)"
End Sub

'=====================================================
' Löschen Daten
'=====================================================

Sub Lösch_Daten()
```

```
Worksheets("Auswertung").Activate
Set r1 = Range(Cells(2, 1), Cells(13, 3))
Set r2 = Range(Cells(15, 3), Cells(16, 3))
Set MehrBlockBereich = Union(r1, r2)
MehrBlockBereich.Select
Selection.ClearContents
Range("E1").ClearContents
Range("A2").Select
End Sub
'=====================================================
'        Datenkorrektur
'=====================================================
Sub Korrektur()
    Sheets.Add
    ActiveSheet.Name = "Korrektur"

    Sheets("Auswertung").Select
    Range("A1:B12").Select
    Selection.Copy
    Sheets("Korrektur").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    ActiveWindow.DisplayGridlines = False
    ActiveWindow.DisplayHeadings = False
    Range("A1").Select
      ActiveSheet.Cells(Rows.Count, ActiveCell.Column).Select
        If IsEmpty(ActiveCell) Then
          ActiveCell.End(xlUp).Select
          A = ActiveCell.Row
          B = ActiveCell.Column
        End If
    Range(Cells(A + 1, B), Cells(A + 9, B + 1)).ClearFormats
    Range("A2").Select
    ActiveSheet.ShowDataForm

    Range(Cells(A, B), Cells(1, B + 1)).Select
    Application.CutCopyMode = False
    Selection.Copy
    Sheets("Auswertung").Select
    Range("A1").Select
    ActiveSheet.Paste

    Sheets("Korrektur").Select
    Application.DisplayAlerts = False
    ActiveWindow.SelectedSheets.Delete
    Application.DisplayAlerts = True
    Range("C16").Select
End Sub
'=====================================================
' Hilfe
'=====================================================
Sub Hilfe()
Worksheets("Hilfe").Visible = True
Worksheets("Hilfe").Select
Range("H20").Select
End Sub
'=====================================================
' Hilfe verlassen
```

```
'=====================================================
Sub Hilfe_Aus()
Worksheets("Auswertung").Select
Worksheets("Hilfe").Visible = False

End Sub
'=====================================================
'        Beenden
'=====================================================
Sub Ende()
  Application.Quit
End Sub
```

13.2 Beispielprogramm 2

(modifizierte Form des Beispielprogramms 1)

```
'=====================================================
'  Initialisierung
'=====================================================
Sub Auto_open()
  Worksheets("Hilfe").Visible = False
  Worksheets("Rdaten").Visible = False
  Sheets("Eingangsblatt").Select
  Anfang.cmdEingabe.Enabled = False
  Anfang.cmdMwSt.Enabled = False
  Anfang.cmdKorrektur.Enabled = False
  Anfang.cmdLöschen.Enabled = False
  Anfang.txtKunde.Text = ""
  ActiveSheet.Protect DrawingObjects:=True, Contents:=True
End Sub
'=====================================================
'  Dateneingabe
'=====================================================
Private Sub cmdEingabe_Click()
 Application.ScreenUpdating = False
  Worksheets("RDaten").Activate
   ActiveWindow.DisplayZeros = False
   Worksheets("RDaten").Range("B2:B13").Style = "Currency"

   With Worksheets("RDaten").Range("C2:C12")
      .Formula = "=B2+B2*$B$15/100"
      .Style = "Currency"
   End With
    Worksheets("RDaten").Range("A2").Select
    Application.DisplayAlerts = False
     ActiveSheet.ShowDataForm
    Application.DisplayAlerts = True
    Worksheets("RDaten").Range("C15").Formula = _
     "=sum(C2:C12)*$B$15/100"
    Worksheets("RDaten").Range("C16").Formula = "=sum(C2:C12)"
    Worksheets("RDaten").Range("C2:C12").Select
    For Each Zelle In Selection
       If Zelle.Value = 0 Then
         Zelle.Value = ""
       End If
    Next
  Application.ScreenUpdating = True
  Worksheets("Eingangsblatt").Activate
  Anfang.cmdKorrektur.Enabled = True
  Anfang.cmdLöschen.Enabled = True

End Sub
'=====================================================
'  Datenkorrektur
'=====================================================
Private Sub cmdKorrektur_Click()
    Dim A
    Dim B
    Sheets.Add
    ActiveSheet.Name = "Korrektur"
    Worksheets("Rdaten").Visible = True
    Sheets("Rdaten").Select
    ActiveSheet.Range("A1:B12").Select
    Selection.Copy
    Sheets("Korrektur").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    ActiveWindow.DisplayGridlines = False
    ActiveWindow.DisplayHeadings = False
    With ActiveSheet
```

```
        .Range("A1").Select
        .Cells(Rows.Count, ActiveCell.Column).Select
          If IsEmpty(ActiveCell) Then
            ActiveCell.End(xlUp).Select
            A = ActiveCell.Row
            B = ActiveCell.Column
          End If
      End With

      With Worksheets(1)
        .Range(.Cells(A + 1, B), .Cells(A + 9, B + 1)).ClearFormats
        .Range("A2").Select
        .ShowDataForm
        .Range(.Cells(A, B), .Cells(1, B + 1)).Select
      End With
          Application.CutCopyMode = False
          Selection.Copy

      Sheets("Rdaten").Select
      ActiveSheet.Range("A1").Select
      ActiveSheet.Paste

      Sheets("Korrektur").Select
      Application.DisplayAlerts = False
      ActiveWindow.SelectedSheets.Delete
      Application.DisplayAlerts = True
      Worksheets("Rdaten").Visible = False

End Sub
'=======================================================
'  Eingabe Kundenname
'=======================================================
Private Sub cmdKunde_Click()
  Anfang.txtKunde.Text = InputBox("Bitte Kundennamen eingeben:", _
                         "Kunde")
  Anfang.cmdMwSt.Enabled = True
End Sub
'=======================================================
'  Eingabe MwSt-Satz
'=======================================================
Private Sub cmdMwSt_Click()
  Worksheets("RDaten").Range("B15").Value = _
  Val(InputBox("Bitte MwSt-Satz eingeben:", "MwSt-Satz"))
  Anfang.cmdEingabe.Enabled = True
End Sub
'=======================================================
'  Löschen
'=======================================================
Private Sub cmdLöschen_Click()
 Antwort = MsgBox("Das Löschen ist in zwei Formen möglich:" & _
      Chr(13) & Chr(13) & "Komplett löschen = Ja" & Chr(13) & _
      "nur Daten = Nein", 35, "Löschabfrage")

 Select Case Antwort

 Case vbYes
 Application.ScreenUpdating = False
  Worksheets("Rdaten").Visible = True
  Worksheets("Rdaten").Select
  Worksheets("Rdaten").Range("A2:C12, C15:C16").ClearContents
  Worksheets("Rdaten").Range("A2").Select
  Worksheets("Rdaten").Visible = False
 Application.ScreenUpdating = True
   Worksheets("Eingangsblatt").Activate
   Anfang.txtKunde.Text = ""
   Anfang.cmdEingabe.Enabled = False
   Anfang.cmdMwSt.Enabled = False
   Anfang.cmdKorrektur.Enabled = False
   Anfang.cmdLöschen.Enabled = False
```

```
 Case vbNo
 Application.ScreenUpdating = False
   Worksheets("Rdaten").Visible = True
   Worksheets("Rdaten").Select
   Worksheets("Rdaten").Range("A2:C12, C15:C16").ClearContents
   Worksheets("Rdaten").Range("A2").Select
   Worksheets("Rdaten").Visible = False
 Application.ScreenUpdating = True
   Worksheets("Eingangsblatt").Activate

 Case Else
   Worksheets("Eingangsblatt").Activate
 End Select

  Worksheets("Eingangsblatt").Activate
End Sub
'=====================================================
'  Ende
'=====================================================
Private Sub Ende_Click()
  End
End Sub
'=====================================================
'  Aufruf Hilfe - Form
'=====================================================
Private Sub Hilfe_Click()
frmHelp.Show
End Sub
```

13.3 Einfache UserForm 1

```
Sub Anfang()
 Eingabe1.Show
End Sub

Private Sub UserForm_MouseMove(ByVal Button As Integer, _
  ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
  Wert1.SetFocus
End Sub

Private Sub Steuer_Click()
  Worksheets("Tabelle2").Range("A3").Value = Eingabe1.Wert1.Value
End Sub

Private Sub Ende_Click()
 End
End Sub
```

13.4 Einfache UserForm 2

- WORKBOOK_OPEN - PROZEDUR

```
Private Sub Workbook_Open()
 Worksheets(1).Activate
 Range("A2").Activate
 frmArtikel.Show
End Sub
```

- WORKSHEET_ ACTIVATE - Prozedur

```
Private Sub Worksheet_Activate()
 Range("A2").Activate
 frmArtikel.Show
End Sub
```

- Prozeduren der Form

```
Private Sub cmdEintrag_Click()
 If cmbArtikel.Text = "" Or cmbPreis = "" Then
  MsgBox "Artikel und Preis eintragen !"
 Else
  Cells(ActiveCell.Row, 1).Value = cmbArtikel.Text
  With Cells(ActiveCell.Row, 2)
   .Value = cmbPreis.Value
   .Style = "Currency"
  End With
  Cells(ActiveCell.Row + 1, 1).Activate
 End If
End Sub

Private Sub cmdEnde_Click()
 End
End Sub

Private Sub UserForm_Initialize()
With cmbArtikel
  .AddItem "Heft"
  .AddItem "Hefter"
  .AddItem "Bleistift"
  .AddItem "Filzschreiber"
  .AddItem "Ordner"
  .AddItem "Tinte"
End With

With cmbPreis
  .AddItem "1.25"
  .AddItem "2.95"
```

```
  .AddItem "1.15"
  .AddItem "1.98"
  .AddItem "3.75"
  .AddItem "2.30"
End With
End Sub
```

13.5 UserForm 3

- INITIALISIERUNG

```
Private Sub UserForm_Initialize()

  cmbZiel.RowSource = "Daten!A1:A12"

 With cmbPreis
  .ColumnCount = 2
  .RowSource = "Daten!A1:B12"
  .TextColumn = 2
 End With

  cmbPers.RowSource = "Daten!D1:D10"
  txtDatum.Text = Date
  cmdBuch.Visible = False
  cmdLösch.Visible = False
  ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
 If ActiveCell.Row = 1 Then
  txtNr.Text = "0001"
 Else
  txtNr.Text = Cells(ActiveCell.Row, 1).Value + 1
 End If
  Cells(ActiveCell.Row + 1, 1).Activate
  Selection.NumberFormat = "0000"
End Sub
```

- BERECHNUNG GESAMTPREIS

```
Private Sub cmdRechnen_Click()
  If chkBuch.Value = True Then
   txtPreis.Text = Val(cmbPreis.Text) * Val(cmbPers.Value) * 0.95
  Else
   txtPreis.Text = Val(cmbPreis.Text) * Val(cmbPers.Value)
  End If
  cmdBuch.Visible = True
  cmdLösch.Visible = True
End Sub
```

- EINTRAG INS TABELLENBLATT

```
Private Sub cmdBuch_Click()
 Dim Zeile As Integer, Spalte As Integer
 Zeile = ActiveCell.Row
 Spalte = ActiveCell.Column
 ActiveCell.Value = txtNr.Text
 Selection.NumberFormat = "0000"
 Cells(Zeile, Spalte + 1).Value = CDate(txtDatum.Text)
 Cells(Zeile, Spalte + 2).Value = cmbZiel.Text
 Cells(Zeile, Spalte + 3).Value = CCur(cmbPreis.Text)
 Cells(Zeile, Spalte + 4).Value = cmbPers.Text
 Select Case chkBuch.Value
   Case True
    Cells(Zeile, Spalte + 5).Value = "Ja"
   Case False
    Cells(Zeile, Spalte + 5).Value = ""
 End Select
 Cells(Zeile, Spalte + 6).Value = CCur(txtPreis.Text)
End Sub
```

- LÖSCHEN DES FORMULARS

```
Private Sub cmdLösch_Click()
 cmdBuch.Visible = False
```

```
 ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
 If ActiveCell.Row = 1 Then
  txtNr.Text = "0001"
 Else
  txtNr.Text = Cells(ActiveCell.Row, 1).Value + 1
 End If
 Cells(ActiveCell.Row + 1, 1).Activate
 cmbZiel.Text = ""
 cmbPreis.Text = ""
 cmbPers.Text = ""
 chkBuch.Value = False
 txtPreis.Text = ""
 cmdBuch.Visible = False
 cmdLösch.Visible = False
End Sub
```

- AUFRUF FORM GESAMTSUMME

```
Private Sub cmdGesamt_Click()
 Dim Pwd As String
  frmGesamt.Show
End Sub
```

- BEENDEN

```
Private Sub cmdEnde_Click()
 Range("A1").Activate
 End
End Sub
```

- PASSWORDPRÜFUNG UND SUMMENAUSGABE

```
Private Sub cmdPwd_Click()
  If txtPwd.Text = "Gogolok" Then
   ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Select
   txtSumme.Value = _
   WorksheetFunction.Sum(Range(ActiveCell, Cells(2,
ActiveCell.Column)))
  Else
   MsgBox "Falsche Angabe, bitte wiederholen !", 16, "Fehler"
   txtPwd.Text = ""
   txtPwd.SetFocus

 End If
 End Sub
```

- BEENDEN DER SUMMENAUSGABE

```
Private Sub cmdBack_Click()
  Hide
End Sub
```

14 Schlußbemerkung

Die in dieser Unterlage vorgestellten Techniken der EXCEL – VBA – Programmierung bilden nur die Basis der Arbeit mit EXCEL und VBA. Für die Nutzung der sehr umfangreichen Syntax ist eine lange Erfahrung und regelmäßiger Umgang mit der EXCEL – Programmierung notwendig. Da diese Unterlage die Basis eines zweitätigen Seminars für Anfäger ist, wurde sie möglichst kurz gehalten, auf komplizierte und damit zeitaufwendige Abläufe wurde bewußt verzichtet. Das Material soll als „Appetithäppchen“ für die Teilnehmenden dienen.

Fortgeschrittene Techniken der Arbeit mit EXCEL – VBA sowie hier nicht erwähnte Bereiche, wie die Programmierung von Symbolleisten und Menüs, Pivot-Tabellen-, Diagramm- und Datenbankprogrammierung werden in spätere Seminare (und damit auch Unterlagen) aufgenommen.

15 Literaturliste

Günter Born	Microsoft Office 97 Visual Basic Programmierung Microsoft Press Deutschland 1977
Josef Broukal	Excel schneller, rascher, sicherer mit Visual Basic für Anwendungen Vienna 1996
Matthew Harris	Visual Basic for Applications in 21 Tagen SAMS, München 1997
Reed Jacobson	Programmierung mit Microsoft Excel für Windows 95 Microsoft Press Deutschland 1975
Michael Kofler	VBA-Programmierung mit Excel97 Addison-Wesley 1998
Michael Ortlepp Raine Osenberg	Excel 97 für Windows 95 und Windows NT Sybex-Verlag, Düsseldorf 1997
Dieter Staas	Excel 97 für Anwendungsprogrammierer Carl Hanser Verlag, München 1997