

VBA in Excel

Hans W. Herber

[Wikibooks.org](https://de.wikibooks.org)

27. Juni 2012

On the 28th of April 2012 the contents of the English as well as German Wikibooks and Wikipedia projects were licensed under Creative Commons Attribution-ShareAlike 3.0 Unported license. An URI to this license is given in the list of figures on page 421. If this document is a derived work from the contents of one of these projects and the content was still licensed by the project under this license at the time of derivation this document has to be licensed under the same, a similar or a compatible license, as stated in section 4b of the license. The list of contributors is included in chapter Contributors on page 415. The licenses GPL, LGPL and GFDL are included in chapter Licenses on page 427, since this book and/or parts of it may or may not be licensed under one or more of these licenses, and thus require inclusion of these licenses. The licenses of the figures are given in the list of figures on page 421.

Inhaltsverzeichnis

I.	Grundlegende Programmierkonzepte und Werkzeuge	3
1.	Namenskonventionen	5
1.1.	Wozu Namenskonventionen?	5
1.2.	Die Bestandteile eines Namens	6
1.3.	Die Variablentypen	9
1.4.	Die MS-Forms-Elemente	10
1.5.	Die Konstanten und benutzerdefinierten Typen	12
1.6.	Die Sprungmarken	13
1.7.	Prozeduren und Funktionen	13
1.8.	Kommentare	14
2.	Visual-Basic-Editor	17
2.1.	Der Editor	17

2.2.	Die Einstellungen	18
2.3.	Die Elemente	19
2.4.	Der Objektkatalog	20
II.	Programmiersyntax und dazugehörige Konzepte	21
3.	Prozeduren	23
3.1.	Begriffsbestimmung, Deklaration und Parameter	23
3.2.	Benutzerdefinierte Funktionen (UDF)	26
3.3.	Unterprogramm (Sub)	27
3.4.	Wann sind Funktionen und wann sind Subs einzusetzen?	29
4.	Funktionen	31
4.1.	Arten der Funktionen	31
4.2.	Einsatz von Excel-Funktionen	32
4.3.	Einsatz von VBA-Funktionen	40
4.4.	Einsatz von benutzerdefinierten Funktionen (UDF)	42
5.	Prozeduraufrufe	55
5.1.	Die Aufruf-Syntax	55
5.2.	Die Programmierbeispiele	58
6.	Gültigkeit von Variablen und Konstanten	71
6.1.	Die Gültigkeit:	71
6.2.	Die Beispiele	73

7.	ByRef und ByVal	91
7.1.	Zu ByRef und ByVal	91
7.2.	Die Beispiele	92
8.	Selektieren und Aktivieren	103
8.1.	Selection, muss das sein?	103
8.2.	Worum geht es hier?	104
8.3.	Wieso ist das Selektieren so verbreitet?	107
8.4.	Selektieren und Referenzieren aufgrund unterschiedlichen Denkens?	108
8.5.	Warum soll nicht selektiert werden?	110
8.6.	In welchen Fällen sollte selektiert werden?	111
8.7.	Wie kann ich das Selektieren verhindern?	112
III.	Schleifen und Wenn-Abfragen	117
9.	Schleifen	119
9.1.	For-Schleifen	120
9.2.	Do-Schleifen	124
9.3.	For-Each-Schleifen	127
9.4.	While-Schleifen	128
10.	Wenn-Abfragen	129
10.1.	Einfache Verzweigung (If ... Then)	129
10.2.	Wenn/Dann/Sonst-Verzweigung (If ... Then ... Else)	130

10.3.	Wenn-Dann-SonstWenn-Verzweigung (If..Then..Elsef...Else..)	131
10.4.	Select-Case-Verzweigung	132
10.5.	Inline Verzweigungen mit Iff()	134
10.6.	Inline Verzweigungen mit Choose()	136
10.7.	Wann sollte welche Verzweigung gewählt werden?	137
11.	Kombination von Schleifen und Wenn-Bedingungen	139
11.1.	Erste leere Zelle ermitteln	139
11.2.	UserForm-Optionsfeld nach Tageszeit aktivieren	140
11.3.	Aktiviertes UserForm-Optionsfeld ermitteln	141
12.	Schleifen und Matrizen	143
12.1.	Arrays in VBA	144
12.2.	Eindimensionale vordimensionierte Matrix füllen	147
12.3.	Eindimensionale Matrix mit vorgegebenem Wert dimensionieren und füllen	148
12.4.	Mehrdimensionale Matrix füllen	149
13.	Variablen und Arrays	153
13.1.	Grundlegendes	153
13.2.	Konstanten	157
13.3.	Variablentypen	158
13.4.	Anmerkungen zu den Variablentypen	160

13.5.	Variablendeklaration	168
13.6.	Einsatz von String-Variablen	168
13.7.	Einsatz von Variant-Variablen	170
13.8.	Einsatz von Public-Variablen	172
13.9.	Übergabe von String-Variablen	173
13.10.	Variablen in Funktionen	174
13.11.	Hierarchische Anordnung der Objekttyp-Variablen	176
13.12.	Collections von Objekttyp-Variablen	178
13.13.	Arrays und Feldvariablen	180
14.	Klassenmodule	183
14.1.	Die Module	183
14.2.	Allgemeingültiges Worksheet_Change-Ereignis	185
14.3.	Eine Ereignisprozedur für mehrere CommandButtons	187
14.4.	Ein- und Auslesen einer Kundenliste	188
14.5.	Ereignissteuerung einer Serie von Labels	191
IV.	Weitergehende Programmierkonzepte	195
15.	Code-Optimierung	197
15.1.	Konstanten	197
15.2.	Objektindex	198
15.3.	Direkte Objektzuweisungen	199

15.4.	Selektieren	200
15.5.	Keine eckigen Klammern	200
15.6.	Direkte Referenzierung	201
15.7.	Dimensionierung	202
15.8.	With-Rahmen	203
15.9.	Excel-Funktionen	204
15.10.	Array-Formeln	207
V.	Programmierbeispiele und Prozedurvorlagen	209
16.	Menü- und Symbolleisten	211
16.1.	Grundsätzliches	211
16.2.	Beispiele für das VBA-Handling von CommandBars	214
17.	Leeren und Löschen von Zellen	237
17.1.	Über Dateieigenschaften	237
17.2.	Programmierbeispiele	238
18.	Leeren und Löschen von Zellen	245
18.1.	Löschen aller leeren Zellen einer Spalte	245
18.2.	Löschen der Zeile, wenn Zelle in Spalte A leer ist	246
18.3.	Löschen aller leeren Zeilen	247
18.4.	FehlerZellen leeren	248
18.5.	FehlerZellen löschen	248

18.6.	Löschen aller Zellen in Spalte A mit "hallo" im Text	249
18.7.	Leeren aller Zellen mit gelbem Hintergrund	249
18.8.	Alle leeren Zellen löschen	250
19.	XL4-Makros in VBA verwenden	251
19.1.	Zum Aufruf von XL4-Makros in VBA	251
19.2.	Programmierbeispiele	252
19.3.	Auslesen eines Wertes aus geschlossener Arbeitsmappe	252
19.4.	Auslesen des ANZAHL2-Wertes aus geschlossener Arbeitsmappe	254
19.5.	Auslesen einer Summe aus geschlossener Arbeitsmappe	255
19.6.	Auslesen eines SVERWEIS-Wertes aus geschlossener Arbeitsmappe	256
19.7.	Auslesen einer Tabelle aus geschlossener und Einlesen in neue Arbeitsmappe	257
19.8.	SVERWEIS aus XL4 anwenden	258
19.9.	Namen über XL4 erstellen und ausblenden	260
19.10.	Benannte Formel über XL4 anlegen und aufrufen	261
19.11.	Routine zum Erstellen, Aufrufen und Löschen der Kalenderwochen- Formel	263
19.12.	Druckprogrammierung über XL4-Makros	264
19.13.	Schließen der Arbeitsmappe verhindern	268
19.14.	Arbeitsblattmenüleiste zurücksetzen	269

19.15.	Bedingtes Löschen von Zeilen	270
20.	Textimport	273
20.1.	Import zur Anzeige in MsgBoxes	273
20.2.	Import zur Konvertierung in eine HTML-Seite	276
20.3.	Import zur Anzeige in einem Arbeitsblatt	277
20.4.	Import zur Übernahme in UserForm-Controls	278
21.	Sortieren	283
21.1.	Schnelle VBA-Sortierroutine	283
21.2.	Dialog zur Verzeichnisauswahl	285
21.3.	Auslesen der Dateinamen in einem Verzeichnis	287
21.4.	Sortieren der Dateien eines Verzeichnisses nach Dateiname	288
21.5.	Sortieren der Dateien eines Verzeichnisses nach Dateidatum	289
21.6.	Sortieren der Arbeitsblätter der aktiven Arbeitsmappe	291
21.7.	Sortieren einer Tabelle nach einer benutzerdefinierten Sortierfolge	292
21.8.	Sortieren einer Datums-Tabelle ohne Einsatz der Excel-Sortierung	292
21.9.	Sortieren einer Tabelle nach sechs Sortierkriterien	293
21.10.	Sortieren mit Ae vor Ä und Sch vor S	294
21.11.	Sortieren nach der Häufigkeit des Vorkommens	295
21.12.	Sortieren einschließlich der ausgeblendeten Zeilen	295
21.13.	Sortieren mehrerer Tabellenblattbereiche	296

21.14. Direkter Aufruf des Sortierdialogs	297
21.15. Aufruf des Sortierdialogs unter Einsatz der Sortier-Schaltfläche	297
21.16. Sortieren per Matrixfunktion	298
21.17. Stringfolge sortieren	300
22. Beispiele für Schleifen	305
22.1. Allgemeines / Einleitung	305
22.2. Schleifentypen-Beispiele	307
22.3. Objektbezogene Beispiele	320
23. Rechtschreibprüfung	333
23.1. Die CheckSpelling-Methode	333
23.2. Wort prüfen	335
23.3. Wort auf englisch prüfen	337
23.4. Steuerelement-TextBox prüfen	340
23.5. Zeichnen-TextBox global prüfen	342
23.6. Zeichnen-TextBox einzeln prüfen	344
23.7. Zellbereich prüfen	346
23.8. Gültigkeitsfestlegungen prüfen	347
23.9. UserForm-TextBox prüfen	350
23.10. UserForm-TextBox prüfen	353
23.11. Bei Eingabe Rechtschreibprüfung aufrufen	355

23.12.	Bei Doppelklick Rechtschreibprüfung aufrufen	356
23.13.	Beim Schließen jeder Arbeitsmappe eine Rechtschreibprüfung durchführen	358
VI.	Anhang	361
24.	Weitere Beispiele	363
24.1.	Belegte Zellen bestimmen	363
24.2.	Add-Ins	365
24.3.	Variablentyp bestimmen	366
24.4.	Arbeitsblattexistenz bestimmen	367
24.5.	Tabellenlisten mit Anwenderformular editieren	370
24.6.	Tabellenlistenzeilen scollen	377
24.7.	Exceldaten in XML-Dokument exportieren	382
24.8.	XML-Daten in Excelblatt importieren	384
24.9.	Exceldaten in Access-Datenbank exportieren	388
24.10.	Pivottabelle aus Accessdatenbank erstellen	390
24.11.	Formula Array	392
24.12.	Bedingte Formatierung	396
24.13.	Zellengroße Diagramme in Arbeitsblatt einfügen	398
24.14.	Datensatzkollektion anlegen	401

25. Excel-Links	409
25.1. Deutschsprachige Links	409
25.2. Englischsprachige Links	412
26. Autoren	415
Abbildungsverzeichnis	421
27. Licensen	427
27.1. GNU GENERAL PUBLIC LICENSE	427
27.2. GNU Free Documentation License	436
27.3. GNU Lesser General Public License	443

Teil I.

**Grundlegende
Programmierkonzepte und
Werkzeuge**

1. Namenskonventionen

1.1. Wozu Namenskonventionen?

Eine einheitliche Form der Namensgebung für Variablen, Konstanten und anderer VBA-Komponenten erleichtert es zum einen Entwicklern, den Code des anderen zu verstehen. Zum anderen findet man sich bei einer disziplinierten Namensvergabe auch in seinem eigenen Code besser zurecht. Gänzlich unerlässlich ist die Vereinbarung, wenn ein VBA-Programm im Team erstellt wird.

Microsoft konnte sich bisher nicht entschließen, Namenskonventionen für Excel/VBA festzulegen. Diese Lücke wurde durch Entwickler im englischen Sprachraum gefüllt, indem sie unverbindliche Standards vereinbarten, die sich allerdings bisher international nicht durchsetzen konnten. Es handelt sich hier um eine Kombination aus vereinbarten Kürzeln und beschreibenden (sprechenden) Namen.

Der Verfasser dieser Zusammenfassung stützt sich im Wesentlichen auf die von Graham Keene und James Barnard im Jahre 1996 veröffentlichten Standards, die er an die aktuellen Excel-Versionen angepasst hat.

1.2. Die Bestandteile eines Namens

Der Name besteht aus 3 Teilen: Präfix, Art und Bezug. Der einzige nicht optionale Bestandteil ist die Art. Da sich jedoch in der Regel mehrere Elemente einer Art im Code befinden, wird – um diese unterscheiden zu können – eine Benennung notwendig:

[präfix]Art [Benennung]

Die eckigen Klammern weisen darauf hin, dass es sich bei den Inhalten um optionale Elemente handelt. Die Klammern selbst sind kein Bestandteil des Namens.

Hier drei Beispiele:

Name	Präfix	Art	Benennung
wksKunden		wks	Kunden
mintTeileNo	m	int	TeileNo
gstrKundName	g	str	KundName

Präfix und Art werden in Kleinbuchstaben geschrieben, das erste Zeichen der Benennung als Großbuchstabe. Dies erleichtert die Lesbarkeit des Namens und lenkt den Blick auf die Benennung. In der Benennung selbst wird im Sinne der besseren Lesbarkeit der erste Buchstabe eines jeden Wortes groß geschrieben. Im Sinne der Internationalisierung des VBA-Codes sind generell in den Namen keine Umlaute oder das ß einzusetzen.

Das Präfix

Das Präfix gibt die Art und Gültigkeit der Variablen oder Konstanten an. Hierfür gelten folgende Festlegungen:

- In Subs oder Functions deklarierte Variablen erhalten **kein Präfix**
- Lokal als Static deklarierte Variablen oder Konstanten erhalten das **Präfix s**, also beispielsweise `sintCounter`
- Variablen, die im Deklarationsteil eines Moduls mit einer Dim oder Private-Anweisung deklariert wurden, erhalten das **Präfix m**, also beispielsweise `mcurSumme`
- Global außerhalb von Subs oder Funktionen deklarierte Variablen erhalten das **Präfix g**, also beispielsweise `gdblGesamtSumme`

Die Art

Hier wird die Art der Variablen festgelegt.

Die Excel-Blätter:

Blatt	Art	Beispiel
-------	-----	----------

Arbeitsblatt	wks	wksKunde
Diagramm	cht	chtVerkaeufe
UserForm	frm	frmRechnungHilfe
(XL97/2000) Dia- logblatt (XL5/7)		
Standardmodule	bas	basMain
Klassenmodule	cls	clsMsg
Excel-4-Makro-Blatt	xl4	xl4Bestellung

1.3. Die Variablentypen

VariablentypArt Beispiel BooleanInDim blnSchalter as Boolean CurrencyCurDim curBetrag As Currency DatedatDim datStartDatum As Date DoubledblDim dblPi as Double IntegerintDim intCounter as Integer LonglngDim lngParam as Long ObjectobjDim objGraph as Object SinglelngDim sngParam as Single StringstrDim strUserName as StringType (benutzerdefiniert)typDim typPartRecord As mtPART_RECORD VariantvarDim varEingabe as Variant

Bei Objektlisten wird der Art ein s hinzugefügt. Beispiele:

- Workbook = wkb - Workbooks = wkbs
- Chart = cht - Charts = chts

1.4. Die MS-Forms-Elemente

Objekt	Art	Beispiel
Label	lbl	lblHelpMessage
TextBox	txt	txtLoginName
ComboBox	cbo	cboMonate
ListBox	lst	lstAufstellung
CheckBox	chk	chkAnlage
OptionButton	opt	optJa
ToggleButton	tgl	tglSchalter
CommandButton	cmd	cmdWeiter
TabStrip	tab	tabTexte
MultiPage	mpg	mpgKalender
SpinButton	spn	spnZaehler
ScrollBar	scr	scrLeiste
Image	img	imgStart
RefEdit	ref	refBereich
TreeView	trv	trvVerteilung
ListView	lsv	lsvOrdner

Objekt	Art	Beispiel
Calendar	cal	calAktuell

1.5. Die Konstanten und benutzerdefinierten Typen

Bei den Konstanten weicht man bei VBA von der sonst üblichen Form Großbuchstaben/Unterstriche (Bsp.=NO_WORKSHEET_ERROR) ab. Die Art der Konstanten wird mit con festgelegt, dem möglicherweise ein Präfix (siehe oben) vorangestellt wird. Für die Benennung gelten die oben getroffenen Festlegungen.

Beispiel: **gconFalscherDatenTyp**

Benutzerdefinierte Typen werden mit einem dem Präfix folgenden t kenntlich gemacht, dem das Präfix gemäß den weiter oben gemachten Regeln folgt. Die Benennung erfolgt hier in Großbuchstaben, wobei die einzelnen Wörter durch Unterstriche getrennt werden.

Beispiel: **mtPART_RECORD**

1.6. Die Sprungmarken

Die festgelegten Regeln für den Namenskonvention von Sprungmarken werden hier nicht übernommen, da eine moderne Excel-Programmierung ohne Sprungmarken auskommt. Hier gibt es allerdings eine Ausnahme: Die Fehlerprogrammierung bei auffangbaren Fehlern. Da es die einzige in einer Prozedur vorkommende Sprungmarke ist, bedarf sie keiner besonderen Kennzeichnung. Ihr Name ist im allgemeinen ErrorHandler.

1.7. Prozeduren und Funktionen

Für die Prozedur- und Funktionsnamen gibt es – mit Ausnahme der Ereignisprozeduren – keine Regeln. Im Interesse einer guten Lesbarkeit und schnellen

Abarbeitung des Codes sollte die Länge 20 Zeichen nicht überschreiten. Sie sollten beschreibend und erklärend sein. Jedes Wort beginnt mit einem Großbuchstaben. Gebräuchlich sind die Wortpaare Verb/Gegenstandswort.

Beispiele: **AufrufenDialog**, **SortierenMatrix**, **WechselnBlatt**

Wenn Sie Ereignisse in XL97/2000 programmieren, werden die Prozedurnamen vom VBE festgelegt und sie haben keinen Einfluss darauf. Ausnahmen bilden benutzerdefinierte Ereignisse und Ereignisse zu Elementen, die nicht zu MSForms gehören. Verwenden Sie hier einen beschreibenden Namen, dem ein Unterstrich und ein Hinweis auf die Art des Ereignisses folgt.

1.8. Kommentare

Die Kommentierung des VBA-Codes wird oft vernachlässigt, ist jedoch eine Notwendigkeit. Sie erfordert einen erheblichen Einsatz von Zeit und Energie. Zum einen sollte ein Dritter die Möglichkeit haben, das Programm zu verstehen,

zum anderen wird man selbst – wenn man nach einem Jahr erneut in den Code einsteigen muss – froh über jede Information sein.

Wichtige Elemente des Kommentars sind die Angabe des Autors, des Erstellungs- und letzten Änderungsdatums. Im Weiteren ist die Kommentierungstechnik abhängig von der Art des Code-Aufbaus.

2. Visual-Basic-Editor

2.1. Der Editor

Der Visual-Basic-Editor stellt die Entwicklungsumgebung für die VBA-Programmierung dar. Sie gelangen zum Editor mit der Tastenkombination **Alt+F11**. Im linken Teil des Fensters sehen Sie den Projekt-Explorer mit den zur Zeit geöffneten Projekten, also Arbeitsmappen und AddIns.

2.2. Die Einstellungen

Über das Menü *Extras / Optionen* können Sie Einstellungen für die Arbeit mit dem Editor vornehmen. Hier einige Empfehlungen:

- Register **Editor**

Aktivieren Sie alle Kontrollkästchen mit Ausnahme des ersten (*Automatische Syntaxüberprüfung*, mehr störend als sinnvoll). Wichtig ist die Aktivierung von *Variablendeklaration erforderlich*. Dies zwingt Sie zu einer zumindest ansatzweise ordentlichen Variablendeklaration.

- Register **Editierformat**

Verändern Sie hier nur dann die Voreinstellungen, wenn außer Ihnen niemand in der Entwicklungsumgebung arbeitet, andernfalls wirken sich die Änderungen für Dritte störend aus.

- Register **Allgemein**

Im Rahmen *Unterbrechen bei Fehlern* sollte die Option *Bei nicht verarbeiteten Fehlern* aktiviert sein. Andernfalls kann es im Rahmen von Fehlerroun-ten zu unerwarteten Programmabbrüchen kommen.

- Register **Verankern**
Es sollten alle Kontrollkästchen mit Ausnahme des letzten (*Objektkatalog*) aktiviert sein.

2.3. Die Elemente

Als Programmierelemente (Container für Ihre Programmierungen) stehen Ihnen zur Verfügung:

- **UserForm**
Ein programmierbarer Dialog mit einer Anzahl von eingebauten und anderen, integrierbaren Steuerelementen.
- **Modul** (Standardmodul)
Hier gehören die Prozeduren mit Ausnahme der Ereignisprogrammierung hinein.
- **Neues Klassenmodul**
Es können neue Klassen gebildet werden.

- **Klassenmodule** der Objekte der Arbeitsmappe; diese Module stellen die Container für die Ereignisprogrammierung dar. Dabei handelt es sich um:
 - Diese Arbeitsmappe
 - Tabelle1 etc.
 - evtl. vorhandene UserForms

2.4. Der Objektkatalog

Über den Objektkatalog (aufzurufen mit **F2**) erhalten Sie eine schnelle Übersicht über die Klassen der Bibliotheken und deren Elemente. Wenn Sie eine Klasse oder ein Element markieren, erhalten Sie mit F1 die zugehörige Hilfe-datei.

Teil II.

Programmiersyntax und dazugehörige Konzepte

3. Prozeduren

3.1. Begriffsbestimmung, Deklaration und Parameter

In VBA ist **Prozedur** der Oberbegriff für Funktionen und Unterprogramme.

Die Deklaration der Prozeduren kann erfolgen als:

- **Public**

Auf eine solche Prozedur kann von allen anderen Prozeduren in allen Modulen zugegriffen werden. Bei Verwendung in einem Modul (mit einer *Option*

Private-Anweisung) kann auf die Prozedur nur innerhalb des Projekts zugegriffen werden.

- **Private**

Auf eine solche Prozedur kann nur durch andere Prozeduren aus dem Modul zugegriffen werden, in dem sie deklariert wurde.

- **Static**

Die lokalen Variablen einer solchen Prozedur bleiben zwischen Aufrufen erhalten. Das Attribut *Static* wirkt sich nicht auf Variablen aus, die außerhalb der Prozedur deklariert wurden, auch wenn sie in der Prozedur verwendet werden.

Die Voreinstellung ist **Public**.

Die Festlegung der Parameter kann erfolgen als:

- **Optional**

Schlüsselwort, das angibt, dass ein Argument nicht erforderlich ist. Alle im Anschluss an *Optional* in der Argumentenliste angegebenen Argumente müssen auch optional sein und mit dem Schlüsselwort *Optional* deklariert

werden. *Optional* kann nicht verwendet werden, wenn *ParamArray* verwendet wird.

- **ByVal**

Das Argument wird als Wert übergeben.

- **ByRef**

Das Argument wird als Referenz übergeben.

- **ParamArray**

Ist nur als letztes Argument in *ArgListe* zulässig und gibt an, dass das letzte Element ein als *Optional* deklariertes Datenfeld mit Variant-Elementen ist.

Das Schlüsselwort *ParamArray* erlaubt die Angabe einer variablen Anzahl von Argumenten und darf nicht in Kombination mit den Schlüsselwörtern *ByVal*, *ByRef* oder *Optional* verwendet werden.

3.2. Benutzerdefinierte Funktionen (UDF)

Weitere Informationen: VBA in Excel/ Funktionen¹

Funktionen werden mit oder ohne Parameter aufgerufen und geben Werte zurück. Der Aufruf kann sowohl über andere Funktionen oder Prozeduren als auch über die Eingabe im Arbeitsblatt erfolgen. Sie kann Excel- und VBA-Funktionen integrieren.

Beispiel für eine Funktion:

```
Function Ostern(iYear As Integer)
Dim iDay As Integer
iDay = (((255 - 11 * (iYear Mod 19)) - 21) Mod 30) + 21
Ostern = DateSerial(iYear, 3, 1) + iDay + (iDay > 48) + _
6 - ((iYear + iYear \ 4 + iDay + (iDay > 48) + 1) Mod 7)
End Function
```

Beispiel für den Aufruf aus einer Prozedur heraus:

```
Sub WannIstOstern()  
    MsgBox "Ostersonntag: " & Ostern(2008)  
End Sub
```

Im Arbeitsblatt kann die Funktion durch folgende Eingabe verwendet werden (Jahreszahl in Zelle A1):

```
=ostern(A1)
```

Wichtig: Wenn eine Funktion aus dem Tabellenblatt heraus aufgerufen wird, kann sie bestimmte VBA-Aktionen, z.B. Blattwechsel, nicht ausführen.

3.3. Unterprogramm (Sub)

Ein Unterprogramm wird mit oder ohne Parameter aufgerufen und gibt keine Werte zurück, kann aber übergebene Variablenwerte verändern. Der Aufruf er-

folgt durch andere Prozeduren, nicht jedoch über eine Eingabe im Arbeitsblatt. Sie können Excel- und VBA-Funktionen integrieren.

Wie in anderen BASIC-Dialekten wird ein Unterprogramm durch das Schlüsselwort SUB gekennzeichnet. Es hat sich deshalb auch der Begriff **Sub** (Mehrzahl: Subs) eingebürgert.

Beispiel einer Prozedur mit dem Aufruf eines Unterprogramms:

```
Sub WertEintragen()  
  Dim datStart As Date, datEnd As Date  
  Dim iTage As Integer  
  datStart = DateSerial(Year(Date), 2, 15)  
  datEnd = DateSerial(Year(Date), 12, 11)  
  Call WertErmitteln(datStart, datEnd, iTage)  
  Range("A1").Value = iTage  
End Sub  
  
Sub WertErmitteln(ByVal datStart, ByVal datEnd, ByRef iDiff As  
Integer)  
  iDiff = datEnd - datStart  
End Sub
```


3.4. Wann sind Funktionen und wann sind Subs einzusetzen?

Verwenden Sie Funktionen immer dann, wenn Sie ein Ergebnis in Tabellenblättern als Formel einsetzen möchten oder wenn Sie aus einer Sub heraus Rückgabewerte anfordern möchten. In allen anderen Fällen sollten Sie sich für Subs entscheiden.

4. Funktionen

4.1. Arten der Funktionen

Bestandteil fast jeder – auch einfachsten – Programmierung sind Funktionen. Bei der Excel-/VBA-Programmierung hat man es mit 3 Gruppen von Funktionen zu tun:

- Excel-Funktionen
- VBA-Funktionen
- Benutzerdefinierte Funktionen

4.2. Einsatz von Excel-Funktionen

Funktionen erwarten in der Regel Übergabewerte, auf deren Grundlage sie ihre Berechnungen durchführen und geben die Berechnungsergebnisse zurück. Sie können grundsätzlich sowohl innerhalb von VBA-Programmen verwendet wie auch in Tabellenblättern eingesetzt werden, wobei beim Einsatz von benutzerdefinierten Funktionen in Tabellenblättern Beschränkungen zu beachten sind.

Eine Reihe von Funktionen gibt es sowohl in Excel als auch in VBA. Bei der Wahl des Einsatzes der einen oder anderen muss beachtet werden, dass gleichlautende Excel/VBA-Funktionen zu durchaus unterschiedlichen Ergebnissen führen können. Hier sei exemplarisch auf die Trim-Funktion hingewiesen, die in VBA Leerzeichen am Anfang und Ende einer Zeichenfolge, bei Excel zusätzlich die überzähligen innerhalb eines Strings entfernt.

Grundsätzlich gilt für alle Funktionen, ob eingebaute, über VBA einzutragende oder benutzerdefinierte, dass sie keine Formatierungen transportieren

können. Über Funktionen, die im Tabellenblatt aufgerufen werden, können Sie beispielsweise keine Hintergrundformate oder Schriftattribute festlegen, dazu benötigen Sie eine Sub. Jedoch können Funktionen, die über den VBA Editor ausgeführt werden, solche Änderungen vornehmen.

4.2.1. Verwendung innerhalb von VBA-Prozeduren

Excel-Funktionen müssen in VBA als solche kenntlich gemacht werden, indem man ihnen entweder ein Application oder ein Worksheetfunction voranstellt. Soll die Arbeitsmappe abwärtskompatibel angelegt werden, ist Application zu verwenden, da die Vorgängerversionen kein Worksheetfunction kennen. Allgemein ist die Verwendung von Worksheetfunction zu empfehlen, da bei deren Einsatz zum einen die Elemente (Funktionen) automatisch aufgelistet werden und zum anderen als weitere Unterstützung die jeweilige Argumentenliste angezeigt wird.

Von diesem Prinzip sollte abgewichen werden, wenn beim Rückgabewert der Funktion Fehlerwerte zu erwarten sind. Worksheetfunction liefert statt des Fehlerwertes den beliebigen, zum Programmabbruch führenden Laufzeitfehler 1004.

So funktioniert es nicht:

```
Function IsExistsA(strTxt As String) As Boolean  
Dim var As Variant  
var = WorksheetFunction.Match(strTxt, Columns(1), 0)  
If Not IsError(var) Then IsExistsA = True  
End Function
```

Die Notwendigkeit des Abfangens des Fehlers kann man sich ersparen, indem man statt Worksheetfunction jetzt Application verwendet:

```
Function IsExistsB(strTxt As String) As Boolean  
Dim var As Variant  
var = Application.Match(strTxt, Columns(1), 0)  
If Not IsError(var) Then IsExistsB = True  
End Function
```

4.2.2. Verwendung im Arbeitsblatt

Sie haben die Möglichkeit, Excel-Funktionen oder deren Ergebnisse in einem Arbeitsblatt eintragen zu lassen. Sinnvollerweise werden die Funktionen (Formeln) dann eingetragen, wenn spätere Wertekorrekturen im zu berechnenden Bereich zu einer Neuberechnung in der Ergebniszelle führen sollen.

Der Eintrag eines absoluten Wertes (Summe des Wertebereiches in Spalte A):

```
Sub SumValue()  
    Dim intRow As Integer  
    intRow = Cells(Rows.Count, 1).End(xlUp).Row  
    Cells(intRow + 1, 1).Value = WorksheetFunction.Sum(Range("A1:A" &  
    intRow))  
End Sub
```

Der Eintrag einer Formel (Summe des Wertebereiches in Spalte A):

```
Sub SumFormula()  
    Dim intRow As Integer
```

```
intRow = Cells(Rows.Count, 1).End(xlUp).Row  
Cells(intRow + 1, 1).Formula = "=Sum(A1:A" & intRow & ")"  
End Sub
```

Für den Formeleintrag bieten sich folgende Möglichkeiten:

Formula

Die Formel wird in englischer Schreibweise eingetragen und umfaßt einen absoluten Bereich:

```
Sub AbsoluteFormel()  
    Range("B1").Formula = "=AVERAGE(A1:A20)"  
End Sub
```


FormulaR1C1

Die Formel wird in englischer Schreibweise eingetragen und umfasst einen relativen Bereich:

```
Sub RelativeFormelA()  
  Range("B2").Select  
  Range("B2").FormulaR1C1 = "=AVERAGE(R[-1]C[-1]:R[18]C[-1])"  
End Sub
```

Sie kann auch einen teils absoluten und teils relativen Bereich umfassen:

```
Sub RelativeFormelB()  
  Range("C2").Select  
  Range("C2").FormulaR1C1 = "=AVERAGE(R1C[-1]:R20C[-1])"  
End Sub
```

FormulaLocal

Die Formel wird in deutscher Schreibweise eingetragen und umfasst einen absoluten Bereich:

```
Sub AbsoluteFormelLocal()  
    Range("B1").FormulaLocal = "=MITTELWERT(A1:A20)"  
End Sub
```

FormulaR1C1Local

Die Formel wird in deutscher Schreibweise eingetragen und umfasst einen relativen Bereich:

```
Sub RelativeFormelALocal()  
    Range("B2").Select  
    Range("B2").FormulaR1C1Local =  
        "=MITTELWERT(Z(-1)S(-1):Z(18)S(-1))"  
End Sub
```

Sie kann auch einen teils absoluten und teils relativen Bereich umfassen:

```
Sub RelativeFormelBLocal()  
    Range("C2").Select  
    Range("C2").FormulaR1C1Local = "=MITTELWERT(Z1S(-1):Z20S(-1))"  
End Sub
```

Beachten Sie neben der deutschen Schreibweise auch die veränderten Begriffe für Zeilen/Spalten - R(Z) und C(S) - sowie den Austausch der eckigen gegen die runden Klammern.

Grundsätzlich sollte mit Formula gearbeitet und FormulaLocal gemieden werden.

FormulaArray

Array-Formeln werden ohne die ihnen eigenen geschweiften Klammern eingegeben. Eine FormulaLocal-Entsprechung gibt es hier nicht.

```

Sub ArrayFormel ()
    Range("B3").FormulaArray = _
        "=SUM((D16:D19="&"Hosen"&"))*(E16:E19="&"rot"&"))*F16:F19)"
End Sub

```

Dem FormulaArray-Befehl kommt einige Bedeutung zu, da Array-Berechnungen in VBA ihre Zeit benötigen und es sich in vielen Fällen empfiehlt, temporäre ArrayFormeln in Zellen eintragen zu lassen, um ihre Werte auszulesen.

4.3. Einsatz von VBA-Funktionen

4.3.1. Verwendung innerhalb von VBA-Prozeduren

Beim Einsatz von VBA-Funktionen ist bei geforderter Abwärtskompatibilität Vorsicht geboten. Während die Anzahl der Excel-Formeln seit Jahren im Wesentlichen konstant geblieben ist, trifft dies für VBA-Funktionen nicht zu.

Im Interesse eines möglichst weitverbreiteten VBA-Einsatzes wird die Palette der VBA-Funktionen ständig erweitert.

Der Aufruf einer VBA-Funktion ist einfachst; hier wird das aktuelle Verzeichnis geliefert:

```
Sub PathAct ()  
    MsgBox CurDir  
End Sub
```

Verlangt die Funktion Parameter, erfolgt der Aufruf mit der Parameterübergabe:

```
Sub TypeAct ()  
    MsgBox TypeName(ActiveSheet)  
End Sub
```

4.3.2. Verwendung im Arbeitsblatt

Ergebnisse von VBA-Funktionen können über den Aufruf in benutzerdefinierten Funktionen auch direkt ins Tabellenblatt eingetragen werden:

```
Function UmgebungsVariable()  
    UmgebungsVariable = Environ("Path")  
End Function
```

4.4. Einsatz von benutzerdefinierten Funktionen (UDF)

4.4.1. Verwendung innerhalb von VBA-Prozeduren

Benutzerdefinierte Funktionen werden in aller Regel dann eingesetzt, wenn mehrfach wiederkehrende Berechnungen durchgeführt werden sollen. Wenn

es denn auch nicht verlangt wird, sollten sowohl die Funktionen selbst, deren Parameter sowie die in den Funktionen verwendeten Variablen sauber dimensioniert werden.

Im folgenden Beispiel wird aus einer Prozedur heraus mehrfach eine Funktion zum Gespertrschreiben der Ortsnamen aufgerufen:

```
Sub PLZundOrt ()  
    Dim intRow As Integer  
    intRow = 1  
    Do Until IsEmpty(Cells(intRow, 1))  
        Cells(intRow, 3) = Cells(intRow, 1) & " " & _  
            Gesperrr(Cells(intRow, 2))  
        intRow = intRow + 1  
    Loop  
End Sub  
  
Function Gesperrr(strOrt As String) As String  
    Dim intCounter As Integer  
    Do Until Len(strOrt) > 10  
        For intCounter = Len(strOrt) - 1 To 1 Step -1  
            If Mid(strOrt, intCounter, 1) <> " " Then  
                strOrt = Left(strOrt, intCounter) & " " & _
```

```

    Right(strOrt, Len(strOrt) - intCounter)
End If
Next intCounter
    Gesperrt = strOrt
End Function

```

Hier wird eine benutzerdefinierte Funktion zur Umrechnung von Uhrzeiten in Industriezeiten unter Berücksichtigung einer Pausenzeit eingesetzt:

```

Sub DateToNumber()
    Dim intRow As Integer
    intRow = 10
    Do Until IsEmpty(Cells(intRow, 1))
        Cells(intRow, 2) = IndustrieZeit(Cells(intRow, 1))
        intRow = intRow + 1
    Loop
End Sub

Function IndustrieZeit(dat As Date) As Double
    Dim dblValue As Double
    dblValue = dat * 24

```


4.4.2. Verwendung im Arbeitsblatt

Dimensionieren Sie die Funktions-Parameter entsprechend dem übergebenen Wert, nicht nach dem Range-Objekt, aus dem der Wert übergeben wird. Dies gilt unabhängig davon, ob die Range-Dimensionierung im aktuellen Fall ebenfalls richtige Ergebnisse zulässt. Vorstehendes gilt selbstverständlich nicht für zu übergebende Matrizen (Arrays). Im Falle einer evtl. notwendigen Abwärtskompatibilität ist zu beachten, dass die Vorgängerversionen von Excel 8.0 (97) das Range-Objekt in der Parameter-Dimensionierung nicht akzeptieren; verwenden Sie hier das Object-Objekt.

Selbstverständlich lässt sich über Funktionen keine Cursor auf Reisen schicken, jegliches Selektieren entfällt. In Excel 5.0 und 7.0 ist es zudem auch nicht möglich, simulierte Richtungstastenbewegungen einzusetzen. Der nachfolgende Code führt dort zu einem Fehler:

```

Function GetLastCellValueA(intCol As Integer) As Double
    Dim intRow As Integer
    intRow = Cells(Rows.Count, intCol).End(xlUp).Row
    GetLastCellValueA = Cells(intRow, intCol).Value
End Function

```

In diesen Versionen müssen die Zellen abgeprüft werden, wobei man von UsedRange als Ausgangsposition ausgehen kann:

```

Function GetLastCellValueB(intCol As Integer) As Double
    Dim intRow As Integer, intRowL As Integer
    intRowL = ActiveSheet.UsedRange.Rows.Count
    For intRow = intRowL To 1 Step -1
        If Not IsEmpty(Cells(intRow, intCol)) Then Exit For
    Next intRow
    GetLastCellValueB = Cells(intRow, intCol).Value
End Function

```

Der Versuch, einen gesuchten und gefundenen Zellwert an eine Funktion zu übergeben, führt bei Excel 8.0 und höher zu einem falschen Ergebnis (Leerstring) und bei den Vorgängerversionen zu einem Fehler:

```

Function GetFindCellValue (intCol As Integer, strTxt As String) As
String
    Dim rngFind As Range
    Set rngFind = Columns(intCol).Find(strTxt, lookat:=xlWhole,
LookIn:=xlValues)
    If Not rngFind Is Nothing Then GetFindCellValue = rngFind.Value
End Function

```

Beachten Sie bitte, dass das in diesem Abschnitt geschriebene sich ausschließlich auf benutzerdefinierte Funktionen bezieht, die in ein Tabellenblatt eingetragen werden.

Unter Umständen muss die Adresse der aufrufenden Zelle den Ausgangspunkt für die in der benutzerdefinierten Funktion ablaufenden Berechnungen bilden. Nur beim Eingabezeitpunkt richtige Ergebnisse bringt hier die Festlegung mit ActiveCell, denn bei irgendeiner Eingabe in eine andere Zelle ist dies die aktive Zelle.

Falsche Verankerung:

```
Function MyValueA(intOffset As Integer) As Variant  
Application.Volatile  
MyValueA = ActiveCell.Offset(0, intOffset).Value  
End Function
```

Richtige Verankerung:

```
Function MyValueB(intOffset As Integer) As Variant  
Application.Volatile  
MyValueB = Application.Caller.Offset(0, intOffset).Value  
End Function
```

Die korrekte Zuweisung erfolgt über Application.Caller.

Benutzerdefinierte Funktionen berechnen sich auch bei eingeschaltete automatischer Berechnung nicht von selbst. Wünscht man eine Berechnung bei jeder Zelleingabe, ist den Funktionen ein Application.Volatile voranzustellen. Mit dieser Anweisung sollte vorsichtig umgegangen werden, denn sie kann Berechnungsabläufe extrem verzögern. In Arbeitsmappen, mit denen ständig

abrufbare Funktionen bereitgestellt werden - bspw. in der Personl.xls - ist sie konsequent zu meiden.

4.4.3. Übergabe von Bereichen

In benutzerdefinierten Funktionen können -neben Werten- auch ein oder mehrere Zellbereiche übergeben werden. So wie man z.B. der eingebauten Funktion =SUMME(D1:D33) mit D1:D33 einen Bereich übergibt, so kann auch einer Benutzerdefinierten Funktion ein Bereich übergeben werden. Der einzige Unterschied hier ist, dass ein Bereich von zunächst unbekannter Größe ausgewertet werden muss.

Das folgende Beispiel zeigt eine Funktion, die einen Bereich als Argument entgegen nimmt und die Beträge des angegebenen Bereichs aufsummiert:

```
Public Function SummeBetrag(Bereich As Excel.Range) As Double  
Dim Zelle As Excel.Range  
For Each Zelle In Bereich.Cells
```

```

' Enthält die Zelle eine Zahl?
If IsNumeric(Zelle.Value) Then
  ' Nur bearbeiten, falls Zahl:
  SummeBetrag = SummeBetrag + Abs(Zelle.Value)
End If
Next Zelle
End Function

```

Die For-Each Schleife geht dabei den markierten Bereich von links nach rechts und dann von oben nach unten durch. Wäre der Bereich A1:B2 markiert worden, würde die Summe in der Reihenfolge $A1 + B1 + A2 + B2$ berechnet.

Manchmal möchte man einen Bereich spaltenweise durchlaufen. In diesem Beispiel bringt dies keinen Vorteil, aber man kann dazu die Spalteneigenschaft des Range-Objekts nutzen:

```

Public Function SummeBetrag(Bereich As Excel.Range) As Double
  Dim Zelle As Excel.Range
  Dim Spalte As Excel.Range
  ' Spalten von 1 bis zur letzten Spalte durchlaufen:
  For Each Spalte In Bereich.Columns

```

```

' Oberste bis zur untersten Zelle durchlaufen:
For Each Zelle In Spalte.Cells
' Enthält die Zelle eine Zahl?
If IsNumeric(Zelle.Value) Then
' Betrag addieren:
SummeBetrag = SummeBetrag + Abs(Zelle.Value)
End If
Next Zelle
Next Spalte
End Function

```

Die verschachtelten For-Each Schleifen gehen dabei den markierten Bereich von oben nach unten und dann von rechts nach links durch. Wäre der Bereich A1:B2 markiert worden, würde die Summe in der Reihenfolge $A1 + A2 + B1 + B2$ berechnet.

Auch benutzerdefinierte Funktionen sollten fehlerhafte Bereichsauswahlen erkennen und darauf reagieren. Die drei folgenden Beispiele zeigen, wie man Bereiche überprüft:

Enthält der Bereich mehr als nur eine Zeile?

```

Public Function NurEineZeile(Bereich As Excel.Range) As Boolean
NurEineZeile = (Bereich.Rows.Count > 1)
If Not NurEineZeile Then
    MsgBox "Nur eine Zeile erlaubt"
End If
End Function

```

Enthält der Bereich mehr als eine Spalte?

```

Public Function NurEineSpalte(Bereich As Excel.Range) As Boolean
NurEineSpalte = (Bereich.Columns.Count > 1)
If Not NurEineSpalte Then
    MsgBox "Nur eine Spalte erlaubt"
End If
End Function

```

Ist der Bereich quadratisch?

```

Public Function NurQuadratischerBereich(Bereich As Excel.Range) As
Boolean
    NurQuadratischerBereich = (Bereich.Rows.Count =

```



```
Bereich.Columns.Count)
```

```
If Not NurQuadratischerBereich Then
```

```
    MsgBox "Quadratischer Bereich erwartet"
```

```
End If
```

```
End Function
```

Wenn eine benutzerdefinierte Funktion zwei Bereiche als Argumente erwartet, kann es erforderlich sein, dass sich diese Bereiche nicht überschneiden. Mit der Funktion `Intersect` wird die Schnittmenge aus beiden Bereichen bestimmt. Falls sich die Bereiche überschneiden, schreibt die Funktion den Fehler `#BEZUG` ins Arbeitsblatt, sonst die Anzahl der Zellen beider Bereiche:

```
Public Function GetrennteBereiche(Bereich1 As Excel.Range, _  
                                Bereich2 As Excel.Range) As Variant
```

```
If Intersect(Bereich1, Bereich2) Is Nothing Then
```

```
    GetrennteBereiche = Bereich1.Cells.Count +
```

```
    Bereich2.Cells.Count
```

```
Else
```

```
    GetrennteBereiche = CVErr(xlErrRef)
```

```
End If
```

```
End Function
```

Wenn ein Fehler in der Zelle erscheinen soll, muss der Datentyp für den Rückgabewert der Funktion Variant sein, denn nur Der Datentyp Variant kann auch Fehlerwerte speichern.

5. Prozeduraufrufe

5.1. Die Aufruf-Syntax

Die Syntax der Aufrufe von VBA-Programmen und -Unterprogrammen mit oder ohne Übergabe von Parametern kann sehr unterschiedlich sein. Achten Sie bitte bei Ihren VBA-Programmierungen darauf, dass Sie Unterprogramme, die sich in der gleichen Arbeitsmappe wie die aufrufende Prozedur befinden, immer mit **Call** aufrufen:

call Unterprogramm

Das vorangestellte **Call** ist optional, sollte aber im Interesse der Übersichtlichkeit des Codes dennoch verwendet werden.

Weichen Sie von dieser Regel nur dann ab, wenn Sie aus Ablaufgründen den Namen der aufzurufenden Unterprozedur variabel halten müssen. Weiter unten folgt hierfür ein Beispiel.

Befindet sich die aufzurufende Prozedur in einem Klassenmodul und der Aufruf erfolgt aus einem anderen Modul, so ist dem Aufruf die Klasse voranzustellen:

```
call Tabelle1.Unterprogramm
```

Als **Private** deklarierte Funktionen können nicht aufgerufen werden.

Prozeduren in anderen Arbeitsmappen oder Anwendungen werden mit **Run** gestartet, wobei der Makroname zusammen mit dem Namen des Container-Dokuments als String übergeben wird:

```
Run "'Mappe1'!MeinMakro"
```

Hierbei ist zu beachten:

- Dateinamen mit Leerzeichen müssen im Run-Aufruf in Apostrophs gesetzt werden
- Die mit Run aufgerufene Arbeitsmappe wird - wenn nicht geöffnet - im aktuellen Verzeichnis (CurDir) gesucht. Nicht machbar ist:

```
Run "'c:\mappe1.xls'!Meldung"
```

5.2. Die Programmierbeispiele

5.2.1. Aufruf eines Makros in der aktuellen Arbeitsmappe ohne Parameterübergabe

Das aufzurufende Unterprogramm befindet sich in einem Standardmodul der aufrufenden Arbeitsmappe.

- Prozedur: CallSimple
- Art: Sub
- Modul: Standardmodul
- Zweck: Unterprogramm aufrufen
- Ablaufbeschreibung:
 - Makroaufruf
 - Code:

```
Sub CallSimple ()
```

```
MsgBox "Ein normaler Aufruf!"  
End Sub
```

5.2.2. Aufruf einer Funktion in der aktuellen Arbeitsmappe mit Parameterübergabe

- Prozedur: CallFunction
- Art: Sub
- Modul: Standardmodul
- weck: Funktion mit Parameter aufrufen und Funktionsergebnis melden
- Ablaufbeschreibung:
 - Meldung eines von einer Funktion ermittelten Wertes
- Code:

```
Sub CallFunction()  
    MsgBox "Anzahl der Punkte der Schaltfläche: " & vbLf & _  
        CStr(GetPixel(ActiveSheet.Buttons(Application Caller)))  
End Sub
```

5.2.3. Aufruf eines Makros in einer anderen Arbeitsmappe ohne Parameterübergabe

- Prozedur: CallWkbA
- Art: Sub
- Modul: Standardmodul
- Zweck: Makro einer anderen Arbeitsmappe ohne Parameter aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Arbeitsmappename an String-Variable übergeben
 - Fehlerroutine starten
 - Arbeitsmappe an Objektvariable übergeben
 - Fehlerroutine beenden
 - Wenn die Arbeitsmappe nicht geöffnet ist...
 - Negativmeldung
 - Sonst...
 - Makro in anderer Arbeitsmappe starten
- Code:


```

Sub CallWkBA()
    Dim sFile As String
    Dim wkb As Workbook
    sFile = "'vb07_test.xls'"
    On Error Resume Next
    Set wkb = Workbooks(sFile)
    On Error GoTo 0
    If wkb Is Nothing Then
        MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht
gefunden!"
    Else
        Run sFile & "!Meldung"
    End If
End Sub

```

5.2.4. Aufruf einer Funktion in einer anderen Arbeitsmappe mit Parameterübergabe

- Prozedur: CallWkBB
- Art: Sub

- Modul: Standardmodul
- Zweck: Funktion einer anderen Arbeitsmappe mit Parameter aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Arbeitsmappename an String-Variable übergeben
 - Fehlerroutine starten
 - Arbeitsmappe an Objektvariable übergeben
 - Fehlerroutine beenden
 - Wenn die Arbeitsmappe nicht geöffnet ist...
 - Negativmeldung
 - Sonst...
- Funktion in anderer Arbeitsmappe aufrufen und Ergebnis melden
- Code:

```

Sub CallWkbb ()
  Dim sFile As String
  Dim wkbb As Workbook
  sFile = "'vb07_test.xls'"
  On Error Resume Next

```

```

Set wkb = Workbooks(sFile)
On Error GoTo 0
If wkb Is Nothing Then
    MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht
    gefunden!"
Else
    MsgBox Run(sFile & "!CallerName", Application.Caller)
End If
End Sub

```

5.2.5. Aufruf eines Makros in einem Klassenmodul einer anderen Arbeitsmappe

- Prozedur: CallWkbC
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro im Klassenmodul einer anderen Arbeitsmappe aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration

- Arbeitsmappenname an String- Variable übergeben
- Fehlerroutine starten
- Arbeitsmappe an Objektvariable übergeben
- Fehlerroutine beenden
- Wenn die Arbeitsmappe nicht geöffnet ist...
- Negativmeldung
- Sonst...
- Makro in anderer Arbeitsmappe starten
- Code:

```

Sub CallWkbc ()
    Dim sFile As String
    Dim wkb As Workbook
    sFile = "'vb07_test.xls'"
    On Error Resume Next
    Set wkb = Workbooks(sFile)
    On Error GoTo 0
    If wkb Is Nothing Then
        MsgBox "Die Testarbeitsmappe " & sFile & " wurde nicht
gefunden!"
    Else

```

```
Run sFile & "!Tabelle1.CallClassModule"  
End If  
End Sub
```

5.2.6. Word-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallWord
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einem Word-Dokument aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Name des Worddokumentes an String-Variable übergeben
 - Wenn die Datei nicht existiert...
 - Negativmeldung
 - Sonst...
 - Word-Instanz bilden
 - Word-Dokument öffnen

- Word-Makro aufrufen
- Word-Instanz schließen
- Objektvariable zurücksetzen
- Code:

```

Sub CallWord()
    Dim wdApp As Object
    Dim sFile As String
    sFile = ThisWorkbook.Path & "\vb07_WordTest.doc"
    If Dir$(sFile) = "" Then
        MsgBox "Test-Word-Dokument " & sFile & " wurde nicht gefunden!"
    Else
        With CreateObject("Word.Application")
            .documents.Open sFile
            .Run "Project.Modull.WdMeldung"
            .Quit
        End With
    End If
End Sub

```

5.2.7. Access-Makro aus Excel-Arbeitsmappe aufrufen

- Prozedur: CallAccess
- Art: Sub
- Modul: Standardmodul
- Zweck: Ein Makro in einer Access-Datenbank aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Name der Access-Datenbank an String-Variable übergeben
 - Wenn die Datei nicht existiert...
 - Negativmeldung
 - Sonst...
 - Access-Instanz bilden
 - Access-Datenbank öffnen
 - Access-Makro aufrufen
 - Access-Instanz schließen
 - Objektvariable zurücksetzen
- Code:

```

Sub CallAccess()
    Dim accApp As Object
    Dim sFile As String
    ' Pfad, wenn die Access-MDB im gleichen Verzeichnis wie die
    XLS-Datei liegt
    sFile = ThisWorkbook.Path & "\vb07_AccessTest.mdb"
    If Dir(sFile) = "" Then
        Beep
        MsgBox "Access-Datenbank wurde nicht gefunden!"
    Else
        With CreateObject("Access.Application")
            .OpenCurrentDatabase sFile
            .Run "AcMeldung"
            .CloseCurrentDatabase
        End With
    End If
End Sub

```


5.2.8. Aufruf von Prozeduren in der aktuellen Arbeitsmappe mit variablen Makronamen

- Prozedur: CallMacros
- Art: Sub
- Modul: Standardmodul
- Zweck: Makros mit variablen Makronamen aufrufen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Das letzte 6 Zeichen des Namens der aufrufenden Schaltfläche an eine String-Variablen übergeben
 - Meldung, dass jetzt zu dem Makro mit dem in der String-Variablen hinterlegten Namen verzweigt wird
 - Makro mit dem in der String-Variablen hinterlegten Namen aufrufen
- Code:

```
Sub CallMacros()  
Dim sMacro As String
```

```
sMacro = Right(Application.Caller, 6)
MsgBox "Ich verzweige jetzt zu " & sMacro
Run sMacro
End Sub
```

6. Gültigkeit von Variablen und Konstanten

6.1. Die Gültigkeit:

Variablen sind Platzhalter für Zeichenfolgen, Werte und Objekte. Sie können Werte oder Objekte enthalten. Abhängig vom Ort und der Art ihrer Deklaration werden ihre Gültigkeit und die Lebensdauer ihrer Werte festgelegt.

- **Deklaration innerhalb einer Prozedur**

Die Variable hat ihre Gültigkeit ausschließlich für diese Prozedur und kann aus anderen Prozeduren nicht angesprochen werden.

- **Deklaration im Modulkopf**

Die Variable gilt für alle Prozeduren dieses Moduls, eine Weitergabe als Parameter ist nicht notwendig.

- **Deklaration im Modulkopf eines Standardmoduls als *Public***

Die Variable gilt für alle Prozeduren der Arbeitsmappe, soweit das die Prozedur enthaltene Modul nicht als *Private* deklariert ist.

Empfehlenswert ist die grundsätzliche Vermeidung von Public-Variablen und der Verzicht auf Variablen auf Modulebene. Es ist nicht immer einfach zu beurteilen, wann diese öffentlichen Variablen ihren Wert verlieren oder wo er geändert wird. Die sauberste Lösung ist die Deklaration innerhalb der Prozeduren und die Weitergabe als Parameter.

Wenn Sie mit öffentlichen Variablen arbeiten, sollten Sie Ihre Variablenamen gemäß den Programmier-Konventionen vergeben und sie so als öffentlich kennzeichnen. Ein vorangestelltes **g** könnte darauf hinweisen, dass es sich

um eine Public-Variable, ein kleines **m**, dass es sich um eine Variable auf Modulebene handelt.

In den nachfolgenden Beispielen wird Deklaration und Verhalten von Variablen demonstriert.

6.2. Die Beispiele

6.2.1. Deklaration auf Prozedurebene

Eine Variable ist innerhalb einer Prozedur deklariert und nur in dieser Prozedur gültig.

- Prozedur: varA
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration

- Ablaufbeschreibung:
- Variablendeklaration
- Wert an Integer-Variablen übergeben
- Wert melden
- Code:

```
Sub VarA()  
    Dim iValue As Integer  
    iValue = 10 + 5  
    MsgBox "Variablenwert: " & iValue  
End Sub
```

6.2.2. Deklaration auf Modulebene

Eine Variable ist innerhalb eines Moduls in jeder Prozedur gültig und wird im Modulkopf deklariert.

- Prozedur: varB und ProcedureA
- Art: Sub

- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Variablendeklaration im Modulkopf
 - Wert an Double-Variable übergeben
 - Unterprogramm ohne Parameter aufrufen
 - Variablenwert melden
- Code:

```
Dim mdModul As Double  
  
Sub VarB()  
    mdModul = 23 / 14  
    Call ProcedureA  
End Sub  
  
Private Sub ProcedureA()  
    MsgBox "Variablenwert: " & mdModul  
End Sub
```

6.2.3. Statische Variable

Eine Variable ist innerhalb einer Prozedur als statisch deklariert und behält bei neuen Prozeduraufrufen ihren Wert.

- Prozedur: varC
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablen demonstration
- Ablaufbeschreibung:
 - Variablendeklaration
 - Aufrufzähler hochzählen
 - Wert melden
 - Wert hochzählen
- Code:

```
Sub VarC()  
    Static iValue As Integer  
    Static iCount As Integer
```



```
iCount = iCount + 1
MsgBox iCount & ". Aufruf: " & iValue
iValue = iValue + 100
End Sub
```

6.2.4. Public-Variable

Eine Variable ist in der Arbeitsmappe in jedem Modul gültig und im Modulkopf eines Moduls als Public deklariert.

- Prozedur: varD und varE für den Folgeaufruf
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Variablendeklaration im Modulkopf
 - Arbeitsblatt an Objektvariable übergeben
 - Arbeitsblattnamen melden

- Im zweiten Aufruf:
- Wenn die Objekt-Variable nicht initialisiert ist...
- Warnton
- Negativmeldung
- Sonst...
- Arbeitsblattnamen melden
- Code:

```
Public gwksMain As Worksheet
```

```
Sub VarD()  
    Set gwksMain = Worksheets("Tabelle1")  
    MsgBox "Blattname: " & gwksMain.Name  
End Sub
```

```
Sub varE()  
    If gwksMain Is Nothing Then  
        Beep  
        MsgBox "Bitte zuerst über Beispiel D initialisieren!"  
    Else  
        MsgBox "Blattname: " & gwksMain.Name
```

End If
End Sub

6.2.5. Übergabe von Variablen an eine Funktion

Variablen an eine Funktion übergeben und den Rückgabewert melden.

- Prozedur: varF und Funktion GetCbm
- Art: Sub/Funktion
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Variablendeklaration
 - Funktions-Rückgabewert in eine Double-Variable einlesen
 - Ergebnis melden
- Die Funktion:
 - Rückgabewert berechnen
- Code:

```

Sub varF()
    Dim dCbm As Double
    dCbm = GetCbm(3.12, 2.44, 1.58)
    MsgBox "Kubikmeter: " & Format(dCbm, "0.00")
End Sub

Private Function GetCbm( _
    dLength As Double, _
    dWidth As Double, _
    dHeight As Double)
    GetCbm = dLength * dWidth * dHeight
End Function

```

6.2.6. ByRef-Verarbeitung in einem Unterprogramm

Variable ByRef an ein Unterprogramm übergeben und den veränderten Rückgabewert melden.

- Prozedur: varG und Unterprogramm ProcedureB
- Art: Sub

- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Variablendeklaration
 - Variable für Rückgabewert initialisieren
 - Unterprogramm mit Parametern aufrufen
 - Ergebnis melden
- Das Unterprogramm:
 - • Rückgabewert berechnen
- Code:

```

Sub varG ()
  Dim dCbm As Double
  dCbm = 0
  Call ProcedureB(3.12, 2.44, 1.58, dCbm)
  MsgBox "Kubikmeter: " & dCbm
End Sub

Private Sub ProcedureB( _
  ByVal dLength As Double, _

```

```
ByVal dWidth As Double, _  
ByVal dHeight As Double, _  
ByRef dErgebnis As Double)  
dErgebnis = dLength * dWidth * dHeight  
End Sub
```

6.2.7. Übergabe von Variablen an eine andere Arbeitsmappe

Variablen an eine Funktion einer anderen Arbeitsmappe übergeben und den Rückgabewert melden.

- Prozedur: varH und Funktion in anderer Arbeitsmappe
- Art: Sub/Funktion
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Variablendeklaration
 - Pfad und Dateinamen der Test-Arbeitsmappe an String-Variablen übergeben
 - Wenn die Test-Arbeitsmappe nicht gefunden wurde...

- Negativmeldung
- Sonst...
- Bildschirmaktualisierung ausschalten
- Wert an Long-Variable übergeben
- Test-Arbeitsmappe öffnen
- Funktion in der Text-Arbeitsmappe aufrufen und Ergebnis in Long-Variable einlesen
- Test-Arbeitsmappe schließen
- Bildschirmaktualisierung einschalten
- Rückgabewert melden
- Code:

```

Sub varH()
    Dim lValue As Long
    Dim sFile As String
    sFile = ThisWorkbook.Path & "\vb04_test.xls"
    If Dir(sFile) = "" Then
        MsgBox "Die Testdatei " & sFile & " fehlt!"
    Else
        Application.ScreenUpdating = False

```

```

lValue = 12345
Workbooks.Open sFile
lValue = Application.Run("vb04_test.xls!Berechnung", lValue)
ActiveWorkbook.Close savechanges:=False
Application.ScreenUpdating = True
MsgBox "Ergebnis: " & lValue
End If
End Sub

Function Berechnung(lWert As Long)
    Berechnung = lWert * 54321
End Function

```

6.2.8. Variablen füllen und zurücksetzen

Variablenwerte werden belegt und zurückgesetzt.

- Prozedur: varI
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration

- Ablaufbeschreibung:
- Variablen Deklaration
- Aktives Arbeitsblatt an eine Objekt-Variable übergeben
- Schleife bilden
- Array mit Werten füllen
- Meldung mit Arbeitsblattnamen, Array-Inhalt und Wert der Zählvariablen
- Meldung, dass die Werte zurückgesetzt werden
- Objektvariable zurücksetzen
- Array zurücksetzen
- Zählvariable zurücksetzen
- Fehleroutine initialisieren
- Arbeitsblattnamen melden (führt zum Fehler)
- Wert des ersten Datenfeldes melden (leer)
- Wert der Zählvariablen melden (0)
- Prozedur verlassen
- Fehleroutine
- Wenn es sich um die Fehlernummer 91 handelt...
- Meldung mit Fehlernummer und Fehlertext

- Nächste Programmzeile abarbeiten
- Code:

```

Sub varI()
    Dim wks As Worksheet
    Dim arr(1 To 3) As String
    Dim iCounter As Integer
    Set wks = ActiveSheet
    For iCounter = 1 To 3
        arr(iCounter) = Format(DateSerial(1, iCounter, 1), "mmmm")
    Next iCounter
    MsgBox "Name des Objekts Arbeitsblatt:" & vbLf & _
        " " & wks.Name & vbLf & vbLf & _
        "Inhalt des Arrays:" & vbLf & _
        " " & arr(1) & vbLf & _
        " " & arr(2) & vbLf & _
        " " & arr(3) & vbLf & vbLf & _
        "Inhalt der Zählvariablen:" & vbLf & _
        " " & iCounter
    MsgBox "Jetzt werden die Variablen zurückgesetzt!"
    Set wks = Nothing
    Erase arr
    iCounter = 0
    On Error GoTo ERRORHANDLER

```

```
MsgBox wks.Name
MsgBox "Wert des ersten Datenfeldes: " & arr(1)
MsgBox "Wert der Zählvariablen: " & iCounter
Exit Sub
ERRORHANDLER:
If Err = 91 Then
    MsgBox "Fehler Nr. " & Err & ": " & Error
    Resume Next
End If
End Sub
```

6.2.9. Konstanten auf Prozedurebene

Konstante auf Prozedurebene als Endpunkt einer Schleife.

- Prozedur: varJ
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:

- Konstantendeklaration
- Variablendeklaration
- Schleife bilden
- Schleife beenden
- Zähvariable melden
- Code:

```

Sub varJ ()
    Const ciLast As Integer = 100
    Dim iCounter As Integer
    For iCounter = 1 To ciLast
    Next iCounter
    MsgBox "Zähler: " & iCounter
End Sub

```

6.2.10. Public-Konstanten

Public-Konstante für alle Prozeduren der Arbeitsmappe.

- Prozedur: varK
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Konstantendeklaration im Modulkopf
 - Meldung mit der Public-Konstanten
- Code:

```
Public Const gciDecember As Integer = 12
```

```
Sub varK()  
    MsgBox "Monat Dezember hat den Index " & gciDecember  
End Sub
```

6.2.11. Übergabe eines variablen Wertes an eine Konstante

Variabler Wert als Konstante. Gegen Versuche, einen variablen Wert an eine Konstante zu übergeben, wehrt sich VBA vehement. Das Beispiel zeigt eine Möglichkeit, das Problem zu umgehen.

- Prozedur: varL
- Art: Sub
- Modul: Standardmodul
- Zweck: Variablendemonstration
- Ablaufbeschreibung:
 - Konstantendeklaration
 - Meldung mit der variablen Konstanten
 - Code:

```
Sub varL()  
    Const cDay As String = "Day(Now())"  
    MsgBox "Tageskonstante: " & Evaluate(cDay)  
End Sub
```

7. ByRef und ByVal

7.1. Zu ByRef und ByVal

Variablen können an Funktionen oder Unterprogramme übergeben, dort zu Berechnungen verwendet und mit geänderten Werten zurückgegeben werden. Entscheidend hierfür ist das Schlüsselwort der Parameter-Definition des aufnehmenden Unterprogramms.

VBA kennt die Parameterübergaben **ByRef** und **ByVal**. Im ersten Fall - das ist die Standardeinstellung, d.h. wenn keine Vorgabe erfolgt, wird der Parameter als **ByRef** behandelt - wird der Wert des Parameters weiterverarbeitet; ände-

rungen sind auch für das aufrufende Programm wirksam. Im zweiten Fall wird eine Kopie des Parameters übergeben; die Wirksamkeit beschränkt sich auf das aufgerufene Unterprogramm und der Parameter im aufrufenden Programm behält seinen ursprünglichen Wert.

Dies gilt nicht für Objekt-Variablen. Diese behalten auch bei der Verwendung des Schlüsselwortes **ByRef** in der aufrufenden Prozedur ihren ursprünglichen Wert.

7.2. Die Beispiele

7.2.1. Aufruf einer benutzerdefinierten Funktion ohne ByRef/ByVal-Festlegung

Die Funktion errechnet anhand der übergebenen Parameter den Wert und gibt diesen an das aufrufende Programm zurück, wobei die übergebenen Parameter nicht geändert werden.


```

Sub CallFunction()
    Dim dQM As Double
    dQM = fncQM( _
        Range("A2").Value, _
        Range("B2").Value, _
        Range("C2").Value)
    MsgBox "Quadratmeter Außenfläche: " & _
        Format(dQM, "0.000")
End Sub

Private Function fncQM( _
    dLong As Double, dWidth As Double, dHeight As Double)
    fncQM = 2 * (dLong * dWidth + _
        dLong * dHeight + _
        dWidth * dHeight)
End Function

```

7.2.2. Aufruf eines Unterprogramms ohne ByRef/ByVal-Festlegung

Das Unterprogramm wird mit den für die Berechnung notwendigen Parametern und zusätzlich mit einer 0-Wert-Double-Variablen, die als Container für das Berechnungsergebnis dient, aufgerufen. Alle Parameter gelten als **ByRef**, da kein Schlüsselwort verwendet wurde.

```
Sub CallMacro()  
    Dim dQM As Double  
    Call GetQM( _  
        dQM, _  
        Range("A2").Value, _  
        Range("B2").Value, _  
        Range("C2").Value)  
    MsgBox "Quadratmeter Außenfläche: " & _  
        Format(dQM, "0.000")  
End Sub  
  
Private Sub GetQM( _  
    dValue As Double, dLong As Double, _
```

```
dWidth As Double, dHeight As Double)  
dValue = 2 * (dLong * dWidth + _  
dLong * dHeight + _  
dWidth * dHeight)
```

```
End Sub
```

7.2.3. Aufruf mit einer Integer-Variablen bei Anwendung von *ByVal*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```
Sub AufrufA()  
  Dim iRow As Integer, iStart As Integer  
  iRow = 2  
  iStart = iRow  
  Call GetRowA(iRow)  
  MsgBox "Ausgangszeile: " & iStart & _  
    vbCrLf & "Endzeile: " & iRow
```

```

End Sub

Private Sub GetRowA(ByVal iZeile As Integer)
    Do Until IsEmpty(Cells(iZeile, 1))
        iZeile = iZeile + 1
    Loop
End Sub

```

7.2.4. Aufruf mit einer Integer-Variablen bei Anwendung von *ByRef*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, damit auch der Wert der Variablen im aufrufenden Programm.

```

Sub AufrufB()
    Dim iRow As Integer, iStart As Integer
    iRow = 2
    iStart = iRow
    Call GetRowB(iRow)

```

```

MsgBox "Ausgangszeile: " & iStart & _
vbLf & "Endzeile: " & iRow
End Sub

Private Sub GetRowB(ByRef iZeile As Integer)
Do Until IsEmpty(Cells(iZeile, 1))
    iZeile = iZeile + 1
Loop
End Sub

```

7.2.5. Aufruf mit einer String-Variablen bei Anwendung von *ByVal*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```

Sub CallByVal()
Dim sPath As String, sStart As String
sPath = ThisWorkbook.Path

```

```

sStart = sPath
Call GetByVal(sPath)
MsgBox "Vorher: " & sStart & _
      vbCrLf & "Nachher: " & sPath
End Sub

Private Sub GetByVal(ByVal sDir As String)
If Right(sDir, 1) &lt;&gt; "\ " Then
    sDir = sDir & "\ "
End If
End Sub

```

7.2.6. Aufruf mit einer String-Variablen bei Anwendung von *ByRef*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, damit auch der Wert der Variablen im aufrufenden Programm.

```

Sub CallByRef()

```

```

Dim sPath As String, sStart As String
sPath = ThisWorkbook.Path
sStart = sPath
Call GetByRef(sPath)
MsgBox "Vorher: " & sStart & _
vbLf & "Nachher: " & sPath
End Sub

Private Sub GetByRef(ByRef sDir As String)
If Right(sDir, 1) &lt;&gt;&gt; "\" Then
    sDir = sDir & "\"
End If
End Sub

```

7.2.7. Aufruf mit einer Objekt-Variablen bei Anwendung von *ByVal*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.

```

Sub CallObjectA()
    Dim rngA As Range, rngB As Range
    Set rngA = Range("A1:A10")
    Set rngB = rngA
    Call GetObjectA(rngA)
    MsgBox "Vorher: " & rngB.Address(False, False) & _
        vbLf & "Nachher: " & rngA.Address(False, False)
End Sub

Private Sub GetObjectA(ByVal rng As Range)
    Set rng = Range("F1:F10")
End Sub

```

7.2.8. Aufruf mit einer Objekt-Variablen bei Anwendung von *ByRef*

Das Unterprogramm wird mit einer Variablen aufgerufen. Der Wert dieser Variablen verändert sich während des Ablaufs des Unterprogramms, ohne dass sich im aufrufenden Programm der Variablenwert ändert.


```
Sub CallObjectB()  
    Dim rngA As Range, rngB As Range  
    Set rngA = Range("A1:A10")  
    Set rngB = rngA  
    Call GetObjectB(rngA)  
    MsgBox "Vorher: " & rngB.Address(False, False) & _  
        vbCrLf & "Nachher: " & rngA.Address(False, False)  
End Sub  
  
Private Sub GetObjectB(ByRef rng As Range)  
    Set rng = Range("F1:F10")  
End Sub
```


8. Selektieren und Aktivieren

8.1. Selection, muss das sein?

Die nachfolgende Abhandlung mag manchem in der Unterschiedenheit übertrieben erscheinen, dennoch hält der Autor eine klare Position in diesem Thema für angebracht, da das Selektieren und Aktivieren von Trainern und Dozenten auch nach einigen Jahren VBA weiter unterstützt wird und sie in der Regel selbst zu eifrigen Selektierern gehören. Ein kleiner Teil hebt sich wohltuend von der Mehrheit ab. Auch in der Literatur wird aus der Angst heraus, sich

Laien gegenüber nicht verständlich machen zu können, das Thema falsch behandelt.

8.2. Worum geht es hier?

Es gibt in MS Office wie auch im wirklichen Office mehrere Möglichkeiten, ein Objekt (MS Office) oder einen Mitarbeiter (Office) anzusprechen oder ihm Anweisungen zu erteilen. Um einem Mitarbeiter in einer Abteilung eines anderen Werkes die freudige Mitteilung einer Gehaltserhöhung - über die sich sein danebenstehender Kollege gelb ärgert - zu übermitteln, kann man ihm das entweder über die Hauspost mitteilen lassen oder ihn in dem anderen Werk besuchen.

In VBA wäre die erste Vorgehensweise Referenzieren und die zweite Selektieren. Als Code sieht die erste Variante so aus:

Sub Referenzieren ()

```
With Workbooks("Factory.xls").Worksheets("Abteilung").Range("A1")  
    .Value = "Gehaltserhöhung"  
    .Interior.ColorIndex = 3  
    .Font.Bold = True  
With .Offset(1, 0)  
    .Interior.ColorIndex = 6  
    .Font.Bold = False  
End With  
End With  
End Sub
```

Der Selektierer hat, um zum gleichen Ergebnis zu kommen, schon etwas mehr Arbeit:

```
Sub Hingehen()  
    Dim wkb As Workbook  
    Application.ScreenUpdating = False  
    Set wkb = ActiveWorkbook  
    Workbooks("Factory.xls").Activate  
    Worksheets("Abteilung").Select  
    Range("A1").Select  
    With Selection  
        .Value = "Gehaltserhöhung"
```

```
.Interior.ColorIndex = 3
.Font.Bold = True
End With
Range("A2").Select
With Selection
    .Interior.ColorIndex = 6
    .Font.Bold = False
End With
wkb.Activate
Application.ScreenUpdating = True
End Sub
```

Im Bürobeispiel bekommt er für seine Mehrleistung den Zusatznutzen, die Freude des Gehaltserhöhten und den Neid dessen Kollegen live mitzuerleben, bei VBA bleibt es bei der Mehrarbeit.

8.3. Wieso ist das Selektieren so verbreitet?

Dass man kaum Code ohne Selektiererei sieht - hiervon sind viele Code-Beispiele aus dem Hause Microsoft nicht ausgeschlossen - ist vor allem in folgenden Dingen begründet:

- Fast jeder in MS Excel mit VBA Programmierende hat seine ersten VBA-Schritte mit dem Makrorecorder gemacht. Der Recorder ist der Meister des Selektierens und des überflüssigen Codes. Es sei ihm gestattet; er hat keine andere Chance.
- Es erleichtert die Flucht vor abstraktem Denken, indem in die Objekte eine Begrifflichkeit gelegt wird, die nur fiktiv ist.
- Es wird von denen, die VBA vermitteln sollen, eingesetzt, um den Lernenden einen Bezug zu den Objekten zu vermitteln. Dies erleichtert zugegebenermaßen die ersten Schritte in diese Programmiersprache, wirkt sich jedoch später eher als Fluch aus.
- In wesentlich stärkerem Maße als bei anderen Programmiersprachen kommen die Programmierenden aus dem Anwenderbereich und/oder dem der

autodidaktisch Lernenden und besitzen in der Regel keine umfassende Ausbildung in den Grundlagen der Programmierung.

8.4. Selektieren und Referenzieren aufgrund unterschiedlichen Denkens?

Der typischer Gedankengang eines Selektierers:

Wenn ich jetzt in das Arbeitsblatt Tabelle1 der Arbeitsmappe Test1 und dort in Zelle F10 gehe, den dortigen Zellinhalt kopiere, ihn dann in Arbeitsblatt Tabelle2 von Arbeitsmappe Test2 trage und in Zelle B5 ablade, habe ich das Ergebnis, was ich haben möchte. Jetzt kann ich wieder in die Arbeitsmappe zurückgehen, von der aus ich losgegangen bin.

Diese Überlegung schlägt sich bei ihm in folgendem Code nieder:

```
Sub SelektiertKopieren()
```



```

Dim wkb As Workbook
Set wkb = ActiveWorkbook
Worksheets("Test1.xls").Activate
Worksheets("Tabelle1").Select
Range("F10").Select
Selection.Copy
Worksheets("Test2.xls").Activate
Worksheets("Tabelle2").Select
ActiveSheet.Range("B5").Select
ActiveSheet.Paste Destination:=ActiveCell
wkb.Activate
Application.CutCopyMode = False
End Sub

```

Wäre er kein Selektierer, würde er sich sagen, ich kopiere aus Arbeitsmappe Test1, Tabelle1, Zelle F10 nach Arbeitsmappe Test2, Tabelle2, Zelle B5.

So sähe dann sein Code auch aus:

```

Sub ReferenziertKopieren()
    Worksheets("Test1").Worksheets("Tabelle1").Range("F10").Copy -
    Worksheets("Test2").Worksheets("Tabelle2").Range("B5")

```

```
Application.CutCopyMode = False  
End Sub
```

8.5. Warum soll nicht selektiert werden?

- Neben der bekannten Tatsache, dass es sich beim Cursor um keinen Auslauf benötigten Dackel handelt, eher um einen ausgesprochen faulen Hund, der nichts mehr als seine Ruhe liebt, spricht noch folgendes gegen das Selektieren:
- Selektieren macht den Code unübersichtlich. Da an jeder Ecke von Selection gesprochen wird, verliert man leicht den Überblick, was denn nun gerade selektiert ist. Besonders gravierend fällt dies bei der VBA-Bearbeitung von Diagrammen auf.
 - Werden Programme von Dritten weiterbearbeitet, sollte man den nachfolgend damit Beschäftigten die Herumirreirei im Selection-Dschungel ersparen.

- Es wird erheblich mehr Code benötigt. Jede zusätzliche Codezeile ist eine zusätzliche potentielle Fehlerquelle und wirkt sich negativ auf die Performance aus. Die Dateigröße verändert sich nicht entscheidend.
- Der Programmablauf wird unruhig und flackernd. Dies kann nicht in jedem Fall durch Setzen des ScreenUpdating-Modus auf False verhindert werden.

8.6. In welchen Fällen sollte selektiert werden?

Es gibt einige Situationen, in denen Selectieren entweder notwendig oder sinnvoll ist. Verlangt wird es von Excel nur in einer verschwindend geringen Anzahl von Fällen. Um einen zu nennen: Das Fenster ist nur zu fixieren, wenn die Tabelle, für die die Fixierung gelten soll, aktiviert ist. Sinnvoll kann es sein, wenn umfangreicher Code mit Arbeiten an und mit Objekten in zwei Arbeitsblättern befasst ist - beispielsweise einem Quell- und einem Zielblatt, zum Programmstart aber ein drittes das Aktive ist. Um den Code übersichtlich und die Schreibearbeit in Grenzen zu halten, kann man jetzt eines der beiden

Blätter aktivieren und das andere in einen With-Rahmen einbinden. Man erspart sich dadurch die beidseitige Referenzierung.

8.7. Wie kann ich das Selektieren verhindern?

Die Selektiererei lässt sich verhindern durch eine exakte Variablendeklaration und -dimensionierung sowie einer darauf aufbauenden genauen Referenzierung der Objekte.

Im Nachfolgenden einige Beispiele:

Kopieren eines Zellbereiches von einer zur anderen Arbeitsmappe, aufrufen aus einer dritten

```
Sub Kopieren()  
    Dim rngSource As Range, rngTarget As Range  
    Set rngSource =  
    Workbooks("Test1.xls").Worksheets(1).Range("A1:F14")
```

```

Set rngTarget = Workbooks("Test2.xls").Worksheets(2).Range("C16")
    rngSource.Copy rngTarget
End Sub

```

Einfügen einer Grafik in eine zweite Arbeitsmappe

```

Sub BildEinfuegenPositionieren()
    Dim wks As Worksheet
    Dim pct As Picture
    Set wks = Workbooks("Test1.xls").Worksheets(1)
    Set pct = wks.Pictures.Insert("c:\excel\zelle.gif")
    pct.Left = 120
    pct.Top = 150
End Sub

```

In Arbeitsblättern 3 bis 12 je einer Serie von 8 Diagrammen in jedem 2. Diagramm den ersten drei SeriesCollections Trendlinien hinzufügen

```

Sub Aufruf()
    Dim wks As Worksheet
    Dim intCounter As Integer
    For intCounter = 3 To 12

```

```

Call Trendlinie(wks)
Next intCounter
End Sub

Private Sub Trendlinie(wksTarget As Worksheet)
    Dim trdLine As Trendline
    Dim intChart As Integer, intCll As Integer
    For intChart = 1 To 7 Step 2
        With wksTarget.ChartObjects(intChart).Chart
            For intCll = 1 To 3
                Set trdLine =
                    .SeriesCollection(intCll).Trendlines.Add(Type:=xlLinear)
                With trdLine.Border
                    Select Case intCll
                        Case 1
                            .ColorIndex = 5
                            .LineStyle = xlDot
                            .Weight = xlThin
                        Case 2
                            .ColorIndex = 7
                            .LineStyle = xlDot
                            .Weight = xlThin
                        Case 3
                            .ColorIndex = 6
                            .LineStyle = xlDot
                    End Select
                End With
            End With
        Next intChart
    End With

```

```

.Weight = xlThin
End Select
End With
Next intCll
End With
Next intChart
End Sub

```

Bereich im aktiven Blatt filtern und die gefilterten Daten in eine neue Arbeitsmappe kopieren. Am Ende wird die aktive Zelle selektiert, um die Filterauswahl aufzuheben.

```

Sub FilternKopieren()
Dim wkb As Workbook
Set wkb = ActiveWorkbook
Application.ScreenUpdating = False
Range("A1").AutoFilter field:=3, Criteria1:="*2*"
Range("A1").CurrentRegion.SpecialCells(xlCellTypeVisible).Copy
Workbooks.Add
ActiveSheet.Paste Destination:=Range("A1")
Columns.AutoFit
wkb.Activate
ActiveSheet.AutoFilterMode = False

```

```
Application.CutCopyMode = False  
ActiveCell.Select  
End Sub
```


Teil III.

Schleifen und Wenn-Abfragen

9. Schleifen

Siehe auch: ../_Beispiele für Schleifen¹

9.1. For-Schleifen

9.1.1. Einfache For-Schleifen

Einfache For-Schleife zum Eintragen von Zahlen in eine Tabelle

In die erste Spalte des aktiven Arbeitsblattes werden die Ziffern 1 bis 100 eingetragen:

```
Sub EintragenZahlen()  
  Dim intRow As Integer  
  For intRow = 1 To 100  
    Cells(intRow, 1) = intRow  
  Next intRow  
End Sub
```

Einfache For-Schleife zum Eintragen von Wochentagen in eine Tabelle

Als einzige Veränderung zum obigen wird in diesem Beispiel über die Zählfvariable der Wochentag, beginnend beim Sonntag, eingetragen.

```
Sub EintragenWochenTage()  
    Dim intTag As Integer  
    For intTag = 2 To 8  
        Cells(intTag, 1) = Format(intTag, "dddd")  
    Next intTag  
End Sub
```

9.1.2. Einfache For-Schleife mit variablem Ende

For-Schleife zum Eintragen einer zu ermittelnden Anzahl von Tagen

Start oder Ende einer Schleife liegen nicht immer fest und müssen möglicherweise bestimmt werden. Hier wird über die DateSerial-Funktion aus VBA der

letzte Tag des aktuellen Monats bestimmt, um, beginnend bei Zelle E1, die Datumeintragen des aktuellen Monats vorzunehmen.

```
Sub EintragenMonatTage()  
    Dim intTag As Integer  
    For intTag = 1 To Day(DateSerial(Year(Date), Month(Date) + 1, 0))  
        Cells(intTag, 5) = DateSerial(Year(Date), Month(Date), intTag)  
    Next intTag  
End Sub
```

9.1.3. Verschachtelte For-Schleife

Verschachtelte For-Schleife zum Eintragen des aktuellen Kalenderjahres

Die Variablen für Jahr, Monat und Tag werden dimensioniert. Das aktuelle Jahr wird an die Jahresvariable übergeben. Die äussere Schleife führt über die 12 Monate, wobei in Zeile 1 der jeweilige Monatsname eingetragen wird. Die

innere Schleife führt über die Anzahl der Tage des jeweiligen Monats und trägt das jeweilige Datum in die Zellen ein. Zu beachten ist, dass Zeilen- und Schleifenzähler unterschiedliche Werte haben können. Im Beispiel werden die Tage erst ab Zeile 2 eingetragen, also wird der Zeilen- gegenüber dem Schleifenzähler um 1 hochgesetzt.

```
Sub EintragenJahr()  
  Dim intYear As Integer, intMonat As Integer, intTag As Integer  
  intYear = Year(Date)  
  For intMonat = 1 To 12  
    Cells(1, intMonat) = Format(DateSerial(1, intMonat, 1), "mmmm")  
    For intTag = 1 To Day(DateSerial(intYear, intMonat + 1, 0))  
      Cells(intTag + 1, intMonat) = DateSerial(Year(Date),  
intMonat, intTag)  
    Next intTag  
  Next intMonat  
End Sub
```

9.2. Do-Schleifen

9.2.1. Do-Schleifen

Do-Schleifen, ähnlich wie While-Schleifen, wiederholen sich beliebig oft. Die Schleife wird erst durch die Anweisung "Exit Do" beendet, die innerhalb der Do-Schleife z.B.(?) in einer If-Abfrage umgesetzt wird.

In dieser Do-Schleife wird eine Zufallszahl ermittelt. Wenn diese dem Index des aktuellen Monats entspricht, erfolgt eine Ausgabe in einer MsgBox.

```
Sub Zufall()  
    Dim intCounter As Integer, intMonth As Integer  
    Randomize  
    Do  
        intCounter = intCounter + 1  
        intMonth = Int((12 * Rnd) + 1)  
        If intMonth = Month(Date) Then  
            MsgBox "Der aktuelle Monat " & _  
                Format(DateSerial(1, intMonth, 1), "mmmm") & _
```



```

" wurde im " & intCounter & _
". Versuch gefunden!"
Exit Do
End If
Loop
End Sub

```

9.2.2. Do-While-Schleifen

In dieser Do-While-Schleife, startend in Zelle A1, werden die Zellen abwärts geprüft, ob ein Suchbegriff darin vorkommt. Ist die Fundstelle erreicht, wird die Schleife verlassen und eine Meldung ausgegeben

```

Sub SuchenBegriff()
Dim intRow As Integer
intRow = 1
Do While Left(Cells(intRow, 1), 7) <> "Zeile 7"
intRow = intRow + 1
Loop
MsgBox "Suchbegriff wurde in Zelle " & _

```

```
Cells(intRow, 1).Address & " gefunden!"  
End Sub
```

9.2.3. Do-Until-Schleifen

In dieser Do-Until-Schleife wird eine Zählvariable hochgezählt, bis der aktuelle Monat erreicht wird. Die Ausgabe erfolgt in einer Messagebox.

```
Sub PruefenWerte()  
    Dim intCounter As Integer  
    intCounter = 1  
    Do Until Month(DateSerial(Year(Date), intCounter, 1)) = _  
        Month(Date)  
        intCounter = intCounter + 1  
    Loop  
    MsgBox "Der aktuelle Monat ist:" & vbCrLf & _  
        Format(DateSerial(Year(Date), intCounter, 1), "mmmm")  
End Sub
```

9.3. For-Each-Schleifen

Es wird eine Objektvariable für ein Arbeitsblatt angelegt und alle Arbeitsblätter einer Arbeitsmappe werden durchgezählt. Das Ergebnis wird in einer MsgBox ausgegeben.

```
Sub ZaehlenBlaetter()  
  Dim wks As Worksheet  
  Dim intCounter As Integer  
  For Each wks In Worksheets  
    intCounter = intCounter + 1  
  Next wks  
  If intCounter = 1 Then  
    MsgBox "Die aktive Arbeitsmappe hat 1 Arbeitsblatt!"  
  Else  
    MsgBox "Die aktive Arbeitsmappe hat " & _  
      intCounter & " Arbeitsblätter!"  
  End If  
End Sub
```

9.4. While-Schleifen

Beispiel ohne "echte" Funktion, dient lediglich zur Veranschaulichung der While-Schleife. Die Schleife zählt so lange hoch (nach jedem Schritt wird das neue Ergebnis ausgegeben) bis die While-Bed. erfüllt ist. Im Gegensatz zur Do-While-Schleife muss die While-Schleife mit "Wend" (steht für "While-Schleifen Ende") beendet werden! (Siehe auch Unterschied While-Wend / Do-While-Loop²)

```
Sub WhileBsp ()  
  Dim i As Integer  
  i = 0  
  While i <> 3  
    MsgBox "While-Schleife: " & i  
    i = i + 1  
  Wend  
End Sub
```

² <http://bytes.com/topic/visual-basic-net/answers/383247-vb-net-101-difference-while-do-while-loop>

10. Wenn-Abfragen

10.1. Einfache Verzweigung (If... Then)

Wenn es sich beim aktuellen Tag um einen Sonntag handelt, wird eine entsprechende Meldung ausgegeben, wenn nicht, erfolgt keine Aktion.

```
Sub WennSonntagMsg()  
    If Weekday(Date) = 1 Then MsgBox "Heute ist Sonntag"  
End Sub
```

10.2. Wenn/Dann/Sonst-Verzweigung (If ... Then ... Else)

In der Regel werden Wenn-/Dann-Abfragen erstellt, um Verzweigungen zu ermöglichen. In Beispiel 2.2 wird bei WAHR die Sonntagsmeldung, bei FALSCH der aktuelle Wochentag ausgegeben.

```
Sub WennSonntagOderMsg()  
    If Weekday(Date) = 1 Then  
        MsgBox "Heute ist Sonntag"  
    Else  
        MsgBox "Heute ist " & Format(Weekday(Date), "dddd")  
    End If  
End Sub
```

10.3. Wenn-Dann-SonstWenn-Verzweigung (If..Then..ElseIf.. ..Else..)

Über ElseIf können weitere Bedingungen mit entsprechenden Verzweigungen angefügt werden.

```
Sub WennSonntagSonstMsg()  
    If Weekday(Date) = 1 Then  
        MsgBox "Heute ist Sonntag"  
    ElseIf Weekday(Date) = 7 Then  
        MsgBox "Heute ist Samstag"  
    Else  
        MsgBox "Heute ist " & Format(Weekday(Date), "dddd")  
    End If  
End Sub
```

Zweckmäßig ist diese Struktur auch bei der Fehlerprüfung, wenn völlig unterschiedliche Bedingungen geprüft werden sollen:

```
Public Function DiscoEinlass(GeburtsTag As Date) As Boolean
```

```

DiscoEinlass = False

If DateSerial(Year(GeburtsTag) + 18, Month(GeburtsTag),
Day(GeburtsTag)) > Date Then
    MsgBox "Sie sind leider noch nicht volljährig"
ElseIf Year(Date) - Year(GeburtsTag) > 65 Then
    MsgBox "Rentner dürfen hier nicht rein!"
ElseIf Weekday(GeburtsTag, vbSunday) <> 1 Then
    MsgBox "Sie sind kein Sonntagskind und können keine Elfen
sehen"
Else
    DiscoEinlass = True
End If
End Function

```

10.4. Select-Case-Verzweigung

Bei mehr als zwei Bedingungen empfiehlt sich meist - wenn möglich - die Select-Case-Prüfung einzusetzen. Der vorliegende Fall wird eingelesen und danach schrittweise auf seinen Wahrheitsgehalt geprüft.


```

Sub PruefeFallMsg ()
Select Case Weekday(Date)
    Case 1, 7: MsgBox "Heute ist kein Arbeitstag"
    Case 2: MsgBox "Heute ist Montag"
    Case 3: MsgBox "Heute ist Dienstag"
    Case 4: MsgBox "Heute ist Mittwoch"
    Case 5: MsgBox "Heute ist Donnerstag"
    Case 6: MsgBox "Heute ist Freitag"
End Select
End Sub

```

Sehr zweckmäßig ist die Select Anweisung auch, wenn man Optionsfelder in einem Formular (hier mit Objektbezeichner Me angesprochen) auswerten möchte. Dazu dreht man die Vergleichsbedingung um, so dass der konstante Teil des Vergleichs (hier True) hinter der Select Case Anweisung steht:

```

Sub ZeigeOption ()
Select Case True
    Case Me.Option1.Value: MsgBox "Option 1 gewählt"
    Case Me.Option2.Value: MsgBox "Option 2 gewählt"
    Case Me.Option3.Value: MsgBox "Option 3 gewählt"
    Case Me.Option4.Value: MsgBox "Option 4 gewählt"

```

```
Case Else: MsgBox "Nichts gewählt"  
End Select  
End Sub
```

Grundsätzlich sollte der häufigste Fall für eine Verzweigung mit der ersten CASE-Anweisung abgefangen werden, um die Laufzeit bei häufigen Aufrufen zu reduzieren.

10.5. Inline Verzweigungen mit IIf()

Für besonders einfache Fälle gibt es auch die Möglichkeit, Verzweigungen in einer Zeile zu erstellen. Die IIf() Funktion ist dabei das Pendant zur IF..Then..Else..End If Struktur. Die folgende Funktion baut einen Text mit einer IIf()-Funktion zusammen:

```
Public Function GeradeOderUngerade(Zahl As Long) As String  
GeradeOderUngerade = "Die Zahl ist eine "& IIf(Zahl Mod 2 = 0,
```

```
"gerade", "ungerade") & " Zahl"  
End Function
```

Diese Form der Verzweigung hat zwei besondere Merkmale:

- Es muss für beide Antwortmöglichkeiten ein Ergebnis angegeben werden
- Die beiden Teile werden unabhängig vom Ergebnis des Vergleichs immer beide ausgeführt. Dies ist zu beachten, falls Funktionen aufgerufen werden.

Das folgende Beispiel illustriert das Problem:

```
Public Function Division(Dividend As Double, Divisor As Double) As  
Double  
    Division = IIf(Divisor = 0, 0, Dividend / Divisor)  
End Function
```

Eigentlich sollte man im vorhergehenden Beispiel davon ausgehen, dass im Falle einer Division durch 0 (z.B. bei Aufruf von =Division(2,0) in einem Tabellenblatt) in dieser speziellen Funktion eine 0 zurückgegeben wird, statt dass ein Fehler die Ausführung unterbricht. Da aber stets alle Teile der IIf()-

Verzweigung ausgeführt werden, probiert VBA auch die Division durch 0 und die ganze Funktion bricht mit einem Fehler ab.

10.6. Inline Verzweigungen mit Choose()

Das Inline Pendant zur Select Case Struktur ist die Choose() Funktion. Das folgende Beispiel zeigt, wie man in einer Zeile dem Datum einen Wochentag zuordnet:

```
Public Function Wochentag (Datum As Date) As String
    Wochentag = Choose(Weekday (Datum, vbMonday), "Mo", "Di", "Mi",
        "Do", "Fr", "Sa", "So")
End Function
```

Hier gilt wie bei If(), dass alle Ausdrücke von VBA ausgeführt werden, egal wie das Ergebnis des Vergleichs ist.

10.7. Wann sollte welche Verzweigung gewählt werden?

Die vermutlich größte Schwierigkeit besteht, falls die Wahl zwischen `IF..Then..ElseIf` und `Select Case` besteht:

- `Select Case` setzt voraus, dass ein Ausdruck eines Vergleiches mit allen anderen verglichen wird, und der sollte in der Zeile mit `Select Case` auftauchen. Damit eignet es sich beispielsweise zur Abfrage von Optionsfeldern (siehe Beispiel oben), zur Abfrage von Bereichen oder wenn eine Funktion wie `MsgBox` mehr als zwei verschiedene Rückgabewerte hat.
- `If..Then..ElseIf` erlaubt es, völlig unterschiedliche Vergleiche auszuführen. `If..Then..ElseIf` eignet sich beispielsweise für Plausibilitätsabfragen am Anfang einer Funktion. Hier werden die Eingabedaten auf oft völlig unterschiedliche Kriterien geprüft, aber wenn nur eines erfüllt ist, gibt es eine spezielle Fehlermeldung.

11. Kombination von Schleifen und Wenn-Bedingungen

11.1. Erste leere Zelle ermitteln

Es wird zuerst geprüft, ob Zelle A1 einen Wert besitzt. Wenn nein, wird die Prozedur verlassen. Danach wird der Zeilenzähler initialisiert. Es folgt eine Schleife über alle Zellen in Spalte A, bis die erste leere Zelle erreicht wird. Die Adresse der ersten leeren Zelle wird in einer MsgBox ausgegeben.

```

Sub GeheBisLeer()
    Dim intRow As Integer
    If IsEmpty(Range("A1")) Then Exit Sub
    intRow = 1
    Do Until IsEmpty(Cells(intRow, 1))
        intRow = intRow + 1
    Loop
    MsgBox "Letzte Zelle mit Wert: " & _
        Cells(intRow - 1, 1).Address(False, False)
End Sub

```

11.2. UserForm-Optionsfeld nach Tageszeit aktivieren

Über die SelectCase-Anweisung wird die aktuelle Stunde bestimmt und hierüber die Tageszeit bzw. das entsprechende Optionsfeld aktiviert. Die Prozedur kommt in das Klassenmodul der UserForm.

```

Private Sub UserForm_Initialize()
    Select Case Hour(Time)

```



```
Case Is > 18: optAbend.Value = True
Case Is > 12: optMittag.Value = True
Case Is > 6:  optMorgen.Value = True
End Select
End Sub
```

11.3. Aktiviertes UserForm-Optionsfeld ermitteln

Es wird zuerst eine Objektvariable für das Control-Objekt initialisiert. Danach werden alle Controls der UserForm durchlaufen. Treffen die Bedingungen, dass es sich um ein Optionsfeld handelt und dass es aktiviert ist zu, dann wird eine entsprechende Meldung ausgegeben und die Schleife beendet.

```
Private Sub cmdWert_Click ()
Dim cnt As Control
For Each cnt In Controls
If Left(cnt.Name, 3) = "opt" And cnt.Value = True Then
    MsgBox "Optionsfeld " & cnt.Name & " ist aktiviert!"
Exit Sub
```

End If

Next cnt

End Sub

12. Schleifen und Matrizen

Matrizen in VBA werden als Arrays bezeichnet. Grundsätzlich gibt es mehrere Möglichkeiten, ein Array zu erzeugen:

- Über Dim als Datenfeld, z.B. ergibt die Anweisung Dim Matrix(1 To 3, 1 To 3) eine 3×3 -Matrix mit der mathematisch richtigen Indizierung der Zeilen und Spalten jeweils von 1..3
- An eine Variable vom Typ Variant kann ein Array aus einer anderen Variablen zugewiesen werden
- Über die Anweisung array() kann an eine Variable vom Typ Variant ein Array zugewiesen werden, z.B. mit VarID = array(11,12,13); Auf diese Art ist es auch möglich, ein zweidimensionales Array anzulegen, z.B. durch

`Var2D = array(array(11, 12), array(21, 22))`; Arrays höherer Dimensionen lassen sich auf vergleichbare Weise anlegen.

Arrays können auch als Rückgabewert einer benutzerdefinierten Funktion definiert werden. Wenn eine benutzerdefinierte Funktion eine 2×2 -Matrix in ein Tabellenblatt zurückgeben soll, muss auf dem Tabellenblatt zuerst ein Bereich mit 2×2 Zellen markiert werden, dann tippt man die Funktion ein und schließt die Eingabe wie bei einer Matrixformel mit Umschalt+Strg+Eingabe ab.

Das Array lässt sich leider nicht als Konstante (über Const) speichern - weder in einer Prozedur/Funktion noch im Deklarationsteil eines Moduls.

12.1. Arrays in VBA

Das erste Beispiel prüft, ob eine Zahl durch eine Gruppe von anderen Zahlen teilbar ist - falls nicht, wird die Zahl selbst zurückgegeben. Der Vorteil bei die-

ser Schreibweise mit einem array() ist, dass das Programm zu einem späteren Zeitpunkt ohne besondere Kenntnisse des Codes erweitert werden kann, indem man der TeilerListe einfach noch ein paar Zahlen anhängt:

```
Public Function TeilerGefunden(Zahl As Long) As Long  
Dim TeilerListe As Variant ' Liste der Primteiler  
Dim Teiler As Variant ' Schleifenvariable  
  
TeilerListe = Array(2, 3, 5, 7, 11, 13)  
TeilerGefunden = Zahl  
  
For Each Teiler In TeilerListe  
    If Zahl Mod Teiler = 0 Then  
        TeilerGefunden = Teiler  
    Exit Function  
End If  
  
Next Teiler  
End Function
```

Das nächste Beispiel nutzt folgende Eigenschaften in Excel: Tabellenblätter haben nicht nur einen Namen (Eigenschaft .Name), der auf der Registerkarte sichtbar ist, sondern auch einen Objektnamen (Eigenschaft .CodeName), der

nur im Projekt-Explorer des VBA-Editors sichtbar ist und auch dann unverändert bleibt, wenn der Benutzer das Blatt umbenennt. Das deutsche Excel legt diesen Namen (.CodeName) standardmäßig wie den Blattnamen (.Name) an, aber -wie geschrieben- ändert er sich .CodeName nicht mehr bei einer Umbenennung des Blattes.

In diesem Falle enthält die Arbeitsmappe zwei Blätter, die als Objekte mit Tabelle1 und Tabelle2 angesprochen werden können. Die Prozedur bestimmt die Anzahl der benutzten Zellen in jedem Blatt und zeigt sie an:

```
Public Sub BelegungTabellenblätter()  
Dim ListeAllerTabellen As Variant ' Liste aller Tabellen  
Dim Tabelle As Variant ' Schleifenvariable
```

```
ListeAllerTabellen = Array(Tabelle1, Tabelle2) ' Zuweisung des  
Objektarrays
```

```
For Each Tabelle In ListeAllerTabellen  
    MsgBox "Tabelle " & Tabelle.Name & " hat " & _  
        Tabelle.UsedRange.Cells.Count & " belegte Zellen"
```

```
Next Tabelle  
End Sub
```

Dieses Beispiel zeigt also, dass das `array()` auch Objekte aufnehmen kann. Auch hier bietet sich wieder die einfache Möglichkeit, den Code später einfach von Hand zu ergänzen.

12.2. Eindimensionale vordimensionierte Matrix füllen

Eine dimensionierte eindimensionale Matrix wird mit der Zählvariablen gefüllt und danach werden die Werte per `MsgBox` ausgegeben.

```
Sub FuellenMatrixEinfach()  
    Dim arrNumbers(1 To 3) As Integer  
    Dim intCounter As Integer  
    For intCounter = 1 To 3  
        arrNumbers(intCounter) = intCounter  
    End For  
End Sub
```

```

Next intCounter
For intCounter = 1 To UBound(arrNumbers)
    MsgBox arrNumbers(intCounter)
Next intCounter
End Sub

```

12.3. Eindimensionale Matrix mit vorgegebenem Wert dimensionieren und füllen

Die Matrix wird auf die Hälfte der Anzahl der Zeilen der mit A1 verbundenen Zellen dimensioniert. Danach werden die Zellinhalte jeder zweiten Zelle der ersten Spalte in die Matrix eingelesen und über eine MsgBox wieder ausgegeben.

```

Sub FuellenMatrixSingle()
    Dim arrCells() As String
    Dim intCounter As Integer, intCount As Integer, intArr As Integer
    Dim strCell As String

```



```

intCount = Range("A1").CurrentRegion.Rows.Count / 2
ReDim arrCells(1 To intCount)
For intCounter = 1 To intCount * 2 Step 2
    intArr = intArr + 1
    arrCells(intArr) = Cells(intCounter, 1)
Next intCounter
For intCounter = 1 To UBound(arrCells)
    MsgBox arrCells(intCounter)
Next intCounter
End Sub

```

12.4. Mehrdimensionale Matrix füllen

Der mit der Zelle A1 zusammengehörige Bereich wird in eine Matrix ein- und eine einzelne Zelle über MsgBox wieder ausgelesen.

```

Sub FuellenMatrixMulti()
    Dim arrJahr As Variant
    arrJahr = Range("A1").CurrentRegion

```

```
MsgBox arrJahr(3, 2)
End Sub
```

Das folgende Beispiel zeigt, wie man den markierten Bereich im aktiven Tabellenblatt ausliest. Die Funktion geht die Auswahl im Tabellenblatt Zeile für Zeile und dann Spalte für Spalte durch. Jeder gefundene Wert wird in ein Element der Variablen Matrix gespeichert. Diese ist dann der Rückgabewert der Funktion MatrixFüllen():

```
Public Function MatrixFüllen() As Double()
Dim ZeileNr As Long ' Zeilenzähler
Dim SpalteNr As Long ' Spaltenzähler

Dim Matrix() As Double ' Matrix

' Matrix auf Zeilen- und Spaltenzahl der Auswahl bringen
' Dabei soll jeder Index bei 1 beginnen
ReDim Matrix(1 To Selection.Rows.Count, 1 To
Selection.Columns.Count)

' Auswahl zeilenweise lesen
For ZeileNr = 1 To Selection.Rows.Count
```

```

' Auswahl spaltenweise lesen
For SpalteNr = 1 To Selection.Columns.Count
With Selection.Cells(ZeileNr, SpalteNr)
If IsNumeric(.Value) Then
    ' Matrix elementweise füllen
    Matrix(ZeileNr, SpalteNr) = .Value
Else
    ' Fehlermeldung ausgeben
    MsgBox "Zelle " & .Address & " enthält keine
Zahl"
Exit Function
End If
End With
Next SpalteNr
Next ZeileNr

' Rückgabewert der Funktion
MatrixFüllen = Matrix
End Function

```

Die Funktion MatrixFüllen() erstellt die Größe der Matrix anhand der Markierung dynamisch und weist den Inhalt der Matrix dem Rückgabewert der Funktion zu. Zur dynamischen Dimensionierung gehört im Beispiel auch, dass

der Index der Matrix mit 1 beginnend definiert wird (mathematische Notation), ohne diese Angabe würde Excel gewohnheitsmäßig die Indizes bei 0 beginnen lassen. Falls eine Zelle keine Zahl enthält, erscheint eine Fehlermeldung. Leere Zellen werden als 0 interpretiert.

13. Variablen und Arrays

13.1. Grundlegendes

Was sind Variablen?

Variablen sind eine Art von Platzhalter für Zeichenfolgen, Werte und Objekte. So können beispielsweise mehrfach anzuzeigende Meldungen, bei Berechnungen mehrfach einzusetzende Werte oder in einer Schleife anzuspärende Objekte in Variablen gespeichert werden.

Wann sind Variablen einzusetzen?

Der Einsatz von Variablen ist immer dann sinnvoll, wenn das Element mehrfach angesprochen wird. Sinnvoll eingesetzt, beschleunigen Variablen die Ausführung eines VBA-Programms erheblich. Wird das Element im Code nur einmal angesprochen – wie zum Beispiel eine Msg-Meldung – ist das Speichern dieser Zeichenfolge in eine String-Variable überflüssig und verwirrend. Ausnahmen bilden Fälle, in denen auch bei einmaligem Vorkommen die Übersichtlichkeit des Codes verbessert wird. Dies kann beispielsweise bei langen Objektnamen der Fall sein.

Sind Variablen zu deklarieren?

Eine Deklaration der Variablen sollte immer erfolgen. Dazu sollte in der Entwicklungsumgebung im Menü **Extras / Optionen** die **CheckBox Variablendeklaration erforderlich** aktiviert sein. VBA-Anweisungen zur Dimensionierung sind:

- Dim
- In einer Function oder Sub Anweisung. Die Deklaration sollte am Anfang stehen

- Zu Beginn eines (Standard-)Moduls oder Klassenmoduls, ist gleichwertig mit Public Dim
- Private: Am Anfang eines (Standard-)Moduls oder Klassenmoduls, bedeutet Private Dim (nicht zulässig)
- Global entspricht Public, aus Gründen der Abwärtskompatibilität unterstützt

Empfehlenswert ist ein Kommentar in der Zeile vor der Variablen Deklaration oder in der Zeile der Deklaration am Ende, um den Zweck der Variablen zu erklären. Beispiel:

```
Private i As Integer ' Schleifenzähler
```

Wo sind Variablen zu deklarieren?

Variablen, die nur für die Prozedur gelten sollen, sind innerhalb der Prozedur, in der Regel am Prozeduranfang zu deklarieren. Variablen, die außerhalb einer Prozedur deklariert werden, gelten für das ganze Modul, werden sie als Public deklariert, für das gesamte Projekt. Zu einem sauberen Programmierstil gehört

es, Variablen soweit möglich nur auf Prozedurebene zu deklarieren und an Unterprogramme als Parameter zu übergeben.

Sind Variablen zu dimensionieren?

Wenn Variablen als Array deklariert wurden, z.B. Dim MitgliedsNr() As Long können sie entweder mit der Deklaration dimensioniert werden (Dim MitgliedsNr(1001) As Long oder Dim MitgliedsNr(1 To 1000) As Long oder nachträglich mit der ReDim-Anweisung

Sind Objekttyp-Variablen bestimmten Objekten zuzuweisen?

Zur Referenzierung von Objekten durch Variable kann stets der allgemeine Typ Variant (nicht empfehlenswert), als auch der allgemeine Objekttyp Object verwendet werden. Wenn die Bibliothek des Objekts über das Menü 'Extras' 'Verweise' eingebunden ist, kann auch der spezielle Objekttyp deklariert werden. Zu bevorzugen ist immer eine möglichst genaue Deklaration, die Deklaration des spezifischen Objekttyps bietet vor allem diese Vorteile:

- Schnellerer Programmablauf

- Weniger Speicherbedarf als bei Variant
- In der Entwicklungsumgebung werden während der Programmierphase - wenn im obigen Dialog die CheckBox Elemente automatisch auflisten aktiviert ist - beim Eintippen des Punktes nach einem Objektnamen alle Methoden und Eigenschaften automatisch aufgelistet, was Fehler vermeidet und Schreibarbeit erspart.
- Fehlermeldungen schon beim Kompilieren (falls beispielsweise Argumente fehlerhaft sind), genauere Fehlerbeschreibungen

13.2. Konstanten

Konstanten werden hier der Vollständigkeit halber erwähnt. Weisen Sie immer dann, wenn ein Wert vom Programmstart bis zum Programmende unverändert bleibt, diesen einer Konstanten, keiner Variablen zu. Konstanten werden in VBA-Programmen schneller berechnet als Variablen. Konstanten werden generell im Allgemein-Abschnitt von Modulen deklariert, Private-Konstanten in

Klassen- und Standard-, Public-Konstanten nur in Standardmodulen. Beispiel für eine Konstanten-Deklaration:

```
Private Const cintStart As Integer = 5
```

13.3. Variablentypen

Die gebräuchlichen Variablentypen:

Variablentyp	Namenskonvention	Res.Speicherplatz	Kurzbezeichnung	Dezimalstellen
Boolean	bln	16 Bit, 2 Bytes	-	-
Byte		8 Bit, 1 Byte	-	-
Integer	int	16 Bit, 2 Bytes	%	-
Long	lng	32 Bit, 4 Bytes	&	-
Currency	cur		@	32
Single	sng	32 Bit, 4 Bytes	!	8
Double	dbl	64 Bit, 8 Bytes	#	16
Date	dat	64 Bit, 8 Bytes		
String	str		\$	

Object obj 32 Bit, 4 Bytes
Variant var 128 Bit, 16 Bytes

benutzerdefinierter Typ
Objektyp

Variablentyp

Boolean WAHR (-1) oder FALSCH (0)

Byte 0 ... +255

Integer -32.768 ... +32.767

Long -2.147.483.648 ... +2.147.483.647

Currency -922.337.203.685.477,5808 ... +922.337.203.685.477,5807

Single ±3,402823E38 ... ±1,401298E-45 und 0

Double -1.79769313486231E308 bis -4,94065645841247E-324 für negative Werte und von 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte und 0

Date Datum und Zeit

String Zeichenfolgen (Text)

Object Objekte

Variant Alle Typen, Voreinstellung

benutzerdefinierter Typ ein oder mehrere Elemente jeden Datentyps. Der Aufbau wird mit einer Type-Anweisung deklariert

Objektyp Objekte wie Workbook, Range

13.4. Anmerkungen zu den Variablentypen

13.4.1. Boolean

Dieser Datentyp speichert eigentlich nur ein Bit, aus Gründen der Speicherorganisation wird jedoch stets ein Byte belegt. Die Werte von Boolean werden als 8-Bit Zahl dargestellt, wobei nur -1 (= alle Bits gesetzt bei Darstellung der -1 als Zweierkomplement) als WAHR gilt, jeder andere Wert aber als FALSCH. Speziell bei Vergleichen wird das Ergebnis FALSCH als 0 (= kein Bit gesetzt) zurückgegeben.

In Kenntnis dieser Interpretation kann der Programmierer Vergleiche auch direkt auf Zahlenwerte in Long-, Integer- und Byte-Datentypen (bei letzteren setzt der Wert 255 alle Bits) anwenden. Aus Gründen der Lesbarkeit des Codes sollte das aber vermieden werden.

13.4.2. Byte

Bei diesem Variablentyp ist in speziellen Fällen Vorsicht geboten, beispielsweise kann bei

```
For i = 1 To 0 Step -1
```

dieser Schleifenkonstruktion ein Unterlauf-Fehler auftreten, wenn i als Byte dimensioniert wird, weil in der internen Berechnung auch noch -1 berechnet wird. Wird als Endwert der Schleife 1 statt 0 angegeben oder wird beispielsweise der Datentyp Integer für i verwendet, gibt es kein Problem.

13.4.3. Date

Der Typ speichert das Datum in zwei Teilen:

- Vor dem Komma steht die fortlaufende Tagesnummer. Tag 0 dieser Zählung ist der 31.12.1899; Bei der Anzeige wird es in die vom System eingestellte Darstellung von Tag, Monat und Jahr umgerechnet.
- Nach dem Komma stehen die Anteile des Tages. 0,25 steht für 6 Stunden, 0,5 für 12 h usw.

Vom Wert her ist der Inhalt dieses Datentyps nicht von einem Fließkommawert zu unterscheiden. Entsprechend einfach können Tage und Stunden addiert werden, hier einige Beispiele:

- Um zu einem Datum h Stunden zu addieren, rechnet man $\text{Datum} + h/24$
- Um zu einem Datum h Stunden und m Minuten zu addieren, rechnet man $\text{Datum} + h/24 + m/(24*60)$ oder $\text{Datum} + (h + m/60)/24$
- Um zu einem Datum h Stunden und m Minuten und s Sekunden zu addieren, rechnet man $\text{Datum} + (h + (m + s/60)/60)/24$

13.4.4. Currency

Der Datentyp ist ein Festkommaformat mit vier Nachkommastellen. Daher wird er intern wie eine Ganzzahl berechnet. Wenn die Genauigkeit ausreicht, kann mit der Wahl dieses Datentyps gegenüber Single und Double die Berechnung schneller erfolgen. Bei Kettenrechnungen mit langen oder periodischen Dezimalteilen ist allerdings mit einem Genauigkeitsverlust zu rechnen.

13.4.5. String

Der Datentyp speichert Zeichen mit variabler Länge von maximal 231 Zeichen. Für bestimmte Zwecke können auch Strings mit fester Länge sinnvoll sein. Sie können mit einem * definiert werden, Beispiel String mit der festen Länge 3:

```
Public Sub Demo_StringMitFesterLänge ()  
    Dim ZeichenKette As String * 3  
    ZeichenKette = "A"
```

```
MsgBox ">" & ZeichenKette & "<"  
End Sub
```

Bei der Zuweisung von "A" wird der String von links belegt, die übrigen Zeichen werden mit einem Leerzeichen aufgefüllt. Die Strings mit fester Länge unterliegen gewissen Einschränkungen, so können sie max. 216 Zeichen speichern und nicht mit dem Attribut Public in Klassenmodulen verwendet werden.

13.4.6. Benutzerdefinierte Typen

Diese Typen werden aus den Grundtypen mit Hilfe der Type-Anweisung zusammengesetzt. Das folgende Beispiel zeigt, wie die Typdeklaration für komplexe Zahlen aussehen könnte. Neben dem Real- und Imaginärteil wird in dem benutzerdefinierten Typ auch gespeichert, ob die komplexe Zahl in kartesischen Koordinaten (FALSE) oder in Polarkoordinaten (TRUE) abgelegt wurde.

Das Beispiel des komplexen Multiplikationsprogramms `cMult` wurde nur für den Fall ausgeführt, in dem beide Variablen in kartesischen Koordinaten vorliegen.

```
Type Komplex      ' Komplexe Zahl
Re                ' Realteil
Im                ' Imaginärteil
Winkel           ' FALSE = Kartesisch, TRUE = Polar
End Type

' ** Funktion zur Multiplikation zweier komplexer Zahlen
Public Function cMult(a As Komplex, b As Komplex) As Komplex
    If (a.Winkel = b.Winkel) Then
        If Not a.Winkel Then
            ' Beide Zahlen liegen im gleichen Koordinatensystem vor
            ' Multiplikation in kartesischen Koordinaten
            cMult.Re = a.Re * b.Re - a.Im * b.Im
            cMult.Im = a.Im * b.Re + a.Re * b.Im
            cMult.Winkel = a.Winkel
        End If
```

```
End If
End Function
```

Das folgende Beispiel zeigt zwei Möglichkeiten, um die Variablen Faktor1 und Faktor2 mit Werten zu belegen und wie man das Ergebnis der Funktion cMult im weiteren Programmablauf verwenden kann:

```
Public Sub Demo_KomplexeMultiplikation()
    Dim Faktor1 As Komplex ' Erster Faktor
    Dim Faktor2 As Komplex ' Zweiter Faktor
    Dim Ergebnis As Komplex ' Komplexes Produkt

    ' Möglichkeit 1.1: Variable mit Hilfe der With-Anweisung
    belegen
        With Faktor1
            .Re = 2
            .Im = 3
            .Winkel = False
        End With

    ' Möglichkeit 1.2: Direkt belegen
    Faktor2.Re = 5
    Faktor2.Im = 7
```

```

Faktor2.Winkel = False
' Möglichkeit 2.1: Ergebnis einer Variablen vom Typ Komplex
zuweisen
  Ergebnis = cMult(Faktor1, Faktor2)
' Ausgabe ins Direktfenster
  Debug.Print Ergebnis.Re, Ergebnis.Im, Ergebnis.Winkel
' Möglichkeit 2.2: Alle Werte einzeln aus dem Rückgabewert
  der Funktion holen
  With cMult(Ergebnis, Faktor2)
    MsgBox Iif(.Winkel, "R: ", "x-Koordinate: ") & .Re
    MsgBox Iif(.Winkel, "Winkel: ", "y-Koordinate: ") & .Im
  End With
End Sub

```

Der Einfachheit halber wurden die Rückgabewerte mit Debug.Print in das Direktfenster geschrieben.

13.5. Variablendeklaration

Wie schon erwähnt, sind Variablen generell zu deklarieren und zu dimensionieren. Werden sie nicht deklariert oder nicht dimensioniert, handelt es sich beim Programmstart in jedem Fall um den Variablentyp Variant, der zum einen mit 16 Bytes den größten Speicherplatz für sich beansprucht, zum anderen während des Programmablaufes seinen Typ mehrmals wechseln kann, was möglicherweise zu unerwarteten Verhalten und damit Fehlern führen kann. Außerdem benötigen Variant-Variablen erheblich längere Berechnungszeiten als andere.

13.6. Einsatz von String-Variablen

Im nachfolgenden Beispiel wird eine String-Variablen deklariert und zum Finden und Ersetzen einer Zeichenfolge eingesetzt:

```

Sub Ersetzen()
    Dim rngCell As Range
    Dim strText As String
    strText = "Kasse "
    strYear = CStr(Year(Date))
    For Each rngCell In Range("A1:F15")
        If rngCell.Value = strText & Year(Date) - 1 Then
            rngCell.Value = strText & Year(Date)
        End If
    Next rngCell
End Sub

```

Im vorgegebenen Bereich werden alle Zellen darauf überprüft, ob ihr Text aus der Zeichenfolge Kasse und der Jahreszahl des Vorjahres besteht. Wenn ja, wird die Vorjahreszahl durch die aktuelle Jahreszahl ersetzt. String-Variablen sollten mit dem &-Zeichen verknüpft werden. Strings können auch mit + verknüpft werden. Dies funktioniert aber nur zuverlässig, wenn beide Variablen oder Ausdrücke strings sind. Falls ein Ausdruck numerisch ist und der andere ein String, der als Zahl interpretierbar ist, nimmt Excel eine Typumwandlung um und liefert als Ergebnis die algebraische Summe der beiden Ausdrücke. Wenn

in einem Ausdruck & mit + gemischt wird, berechnet VBA zuerst + (und alle anderen algebraischen Operationen wie -*/) dann erst &;

Beispiele:

- Aus "2" + "3" wird "23"
- Aus "2" + 3 wird 5
- Aus "2" & 3 wird "23"
- Aus "2" & 3 + 4 & "5" wird 275
- Aus "2" & 3 & 4 & "5" wird 2345
- Aus "2" + 3 & 4 + "5" wird 59

13.7. Einsatz von Variant-Variablen

Es gibt Fälle, in denen eine Variable ihren Typ ändert oder unterschiedliche Typen entgegennehmen muss. In diesem Fall können Variant-Variablen eingesetzt werden. Dies ist besonders dann notwendig, wenn eine Funktion unterschiedli-

che Datentypen zurückgeben kann, wie z.B. GetOpenFilename. Diese liefert entweder einen String als Pfadangabe oder den booleschen Wert FALSE, wenn in dem von ihr geöffneten Dialog die Schaltfläche 'Abbrechen' betätigt wurde:

```
Sub Oeffnen()  
    Dim varFile As Variant  
    varFile = Application.GetOpenFilename("Excel-Dateien (*.xls),  
    *.xls")  
    If varFile = False Then Exit Sub  
    Workbooks.Open varFile  
End Sub
```

Ein anderes Beispiel ist die Funktion IsMissing, mit der geprüft werden kann, ob einer Funktion ein optionales Argument übergeben wurde:

```
Public Sub EingabeMöglich(Optional Wert As Variant)  
    If IsMissing(Wert) Then  
        MsgBox "Kein Argument übergeben"  
    Else  
        MsgBox Wert  
    End If  
End Sub
```

Falls das übergebene Argument in (Optional Wert As String) geändert wird, funktioniert IsMissing() nicht mehr und das Programm durchläuft immer den Else-Zweig.

13.8. Einsatz von Public-Variablen

Im nachfolgenden Beispiel wird in einem Standardmodul eine Public-String-Variablen deklariert. Diese wird in der Prozedur AufrufenMeldung mit einem Wert belegt; danach wird das Unterprogramm Meldung aufgerufen. Da die Variable außerhalb der Prozeduren deklariert wurde, ist der Wert nicht verlorengegangen und kann weiterverwertet werden.

```
Public strMsg As String  
  
Sub AufrufenMeldung ()  
    strMsg = "Hallo!"  
    Call Meldung  
End Sub
```



```
Sub Meldung()  
    MsgBox strMsg  
End Sub
```

Auch wenn sich die Prozedur Meldung in einem anderen Modul befindet, funktioniert der Aufruf. Erfolgt jedoch die Deklaration mit Dim oder als Private, gilt sie nur für das jeweilige Modul.

13.9. Übergabe von String-Variablen

Eine Vorgehensweise wie im vorhergehenden Beispiel ist zu meiden und eine Übergabe der Variablen als Parameter ist vorzuziehen:

```
Sub AufrufenMeldung()  
    Dim strMsg As String  
    strMsg = "Hallo!"  
    Call Meldung(strMsg)  
End Sub
```

```
Sub Meldung(strMsg As String)  
    MsgBox strMsg  
End Sub
```

13.10. Variablen in Funktionen

Funktionen werden eingesetzt, wenn Werte zurückgeliefert werden müssen. Eine Alternative wäre (neben einer ByRef-Variablenübergabe) der Einsatz von Public-Variablen, die wir ja meiden wollen. Bei den Parametern einer Funktion handelt es sich ebenfalls um Variablen. Der Deklarationsbereich liegt innerhalb der Klammern der Funktion. Diese Parameter müssen beim Aufruf der Funktion - aus einem Tabellenblatt oder aus einer anderen Prozedur - übergeben werden. In der nachfolgenden Funktion wird die Kubatur errechnet:

```
Function Kubatur( _  
    dblLaenge As Double, _  
    dblBreite As Double, _
```

```
dblHoehe As Double) As Double  
    Kubatur = dblLaenge * dblBreite * dblHoehe  
End Function
```

Die Eingabesyntax einer solchen Prozedur in einem Tabellenblatt ist, wenn die Werte in den Zellen A1:C1 stehen:

```
=kubatur(A1;B1;C1)
```

Wird die Funktion aus einer anderen Prozedur zur Weiterverarbeitung aufgerufen, sieht das wie folgt aus:

```
Sub ErrechneGewicht()  
    Dim dblSpezGewicht As Double, dblKubatur As Double  
    dblSpezGewicht = 0.48832  
    dblKubatur = Kubatur(Range("A1"), Range("B1"), Range("C1"))  
    Range("E1").Value = dblKubatur * dblSpezGewicht  
End Sub
```

13.11. Hierarchische Anordnung der Objekttyp-Variablen

Über die Objekttypvariablen kann ein Typengerüst aufgebaut werden, indem die jeweils aktuelle Ebene referenziert wird:

```
Sub NeueSymbolleiste()  
    Dim objCmdBar As CommandBar  
    Dim objPopUp As CommandBarPopup  
    Dim objButton As CommandBarButton  
    Dim intMonth As Integer, intDay As Integer  
    On Error Resume Next  
    Application.CommandBars("Jahr " & Year(Date)).Delete  
    On Error GoTo 0  
    Set objCmdBar = Application.CommandBars.Add("Jahr " & Year(Date),  
msoBarTop)  
    For intMonth = 1 To 12  
        Set objPopUp = objCmdBar.Controls.Add(msoControlPopup)  
        objPopUp.Caption = Format(DateSerial(1, intMonth, 1), "mmmm")  
        For intDay = 1 To Day(DateSerial(Year(Date), intMonth + 1, 0))  
            Set objButton = objPopUp.Controls.Add
```

```

With objButton
    .Caption = Format(DateSerial(Year(Date), intMonth,
intDay), _
        "dd.mm.yy - dddd")
    .OnAction = "MeldenTag"
    .Style = msoButtonCaption
End With
Next intDay
Next intMonth
objCmdBar.Visible = True
End Sub

```

Mit vorstehendem Code wird eine neue Symbolleiste mit dem Namen des aktuellen Jahres angelegt und im Symbolleistenbereich als nächstuntere platziert. Der Leiste wird für jeden Monat ein Menü und diesem Menü wird für jeden Tag eine Schaltfläche hinzugefügt.

Das Auslesen der betätigten Schaltfläche und die Datumsberechnungen erfolgen anhand einer Datumsvariablen:

```

Private Sub MeldenTag ()
Dim datAC As Date

```

```

dataAC = DateSerial(Year(Date), Application.Caller(2),
Application.Caller(1))
Select Case dataAC
    Case Is < Date
        MsgBox Date - dataAC & " Tage vergangen"
    Case Is = Date
        MsgBox "Heute"
    Case Is > Date
        MsgBox "Noch " & dataAC - Date & " Tage"
End Select
End Sub

```

13.12. Collections von Objekttyp-Variablen

Das Objekt UserForm1.Controls stellt alle Steuerelemente dar, die in der UserForm1 enthalten sind. Nicht ganz so einfach ist es, auf alle CheckBoxes dieser UserForm zuzugreifen, um sie über eine Schleife zu bearbeiten, denn die CheckBox ist kein gültiges Objekt, das heißt Controls. Liest man die

CheckBoxes in ein Collection-Objekt ein, lassen Sie sich später problemlos ansprechen und in Schleifen einbinden:

```
Public colCheckBox As New Collection

Private Sub UserForm_Initialize()
    Dim cnt As Control, intMonth As Integer
    For Each cnt In Controls
        If TypeName(cnt) = "CheckBox" Then
            intMonth = intMonth + 1
            colCheckBox.Add cnt
        cnt.Caption = Format(DateSerial(1, intMonth, 1), "mmmm")
    End If
Next cnt
End Sub
```

Das Collection-Objekt wird - damit es seinen Wert nicht verliert - als Public außerhalb einer Prozedur deklariert und im Initialisierungscode der UserForm mit den Einzelobjekten - den 12 CheckBoxes der UserForm - belegt. Beim Klick auf die Schaltfläche Meldung werden alle aktivierten CheckBoxes in einer MsgBox ausgegeben:

```

Private Sub cmdMeldung_Click ()
    Dim intCounter As Integer
    Dim strMsg As String
    strMsg = "Aktiviert:" & vbCrLf
    For intCounter = 1 To 12
        If colChBox(intCounter).Value Then
            strMsg = strMsg & colChBox(intCounter).Caption & vbCrLf
        End If
    Next intCounter
    MsgBox strMsg
End Sub

```

13.13. Arrays und Feldvariablen

Es gibt grundsätzlich zwei Möglichkeiten, Variablen für Matrizen zu schaffen. Entweder man deklariert die Variable als Variant als Variant und weist ihr ein Array zu oder man deklariert sie als Datenfeld. Variant-Variablen können Datenfeldvariablen aufnehmen.

Beispiel

```
Dim Array(1 to 200) as integer  
'Zuweisung von Werten  
Array(1) = 1
```


14. Klassenmodule

14.1. Die Module

Module sind Container für Code und für Variablen. Code ist jede Funktion, die einen oder mehrere Werte zurückgibt oder ein Makro, das keine Werte zurückliefert. Ein Modul ist also ein Container für VBA-Routinen.

Excel/VBA kennt Standard- und Klassenmodule. In Standardmodule wird Code zum allgemeinen Programmablauf hinterlegt, Klassenmodule verwalten Objekte mit ihren Eigenschaften, Methoden und Ereignissen.

In Excel gibt es eine Vielzahl von vordefinierten Klassen, um einige zu nennen:

Workbook

In der Entwicklungsumgebung standardmäßig mit dem Objektnamen DieseArbeitsmappe bzw. ThisWorkbook benannt.

Worksheet

In der Entwicklungsumgebung standardmäßig mit den jeweiligen Arbeitsblattnamen benannt.

Chart

In der Entwicklungsumgebung standardmäßig mit den jeweiligen Chart-Namen benannt.

UserForm

In der Entwicklungsumgebung standardmäßig mit dem jeweiligen UserForm-Namen benannt.

Die vorgenannten eingebauten Excel-Klassen können mit ihren Ereignissen in neue Klassen eingebunden werden. Sinnvoll ist dies beispielsweise, wenn eine Worksheet_Change-Ereignisprozedur allgemeingültig werden, sich also nicht nur auf die Arbeitsmappe beschränken soll, in der sich der Code befindet.

14.2. Allgemeingültiges Worksheet_Change-Ereignis

Hier wird eine dem Workbook-Objekt übergeordnete Klasse, also das Application-Objekt als Ausgangspunkt benötigt. In der Entwicklungsumgebung wird über das Menü Einfügen ein neues Klassenmodul erstellt. Der Name des neuen Klassenmoduls kann mit dem Aufruf der Eigenschaften mit der F4-Taste geändert werden (in diesem Fall 'clsApp').

In das Klassenmodul wird zum einen eine Public-Variable für das Ereignis des Application-Objekts und zum anderen der zugehörige Ereigniscode eingetragen:

```
Public WithEvents App As Application

Private Sub App_SheetChange( _
    ByVal Sh As Object, _
    ByVal Target As Range)
    MsgBox "Zelle " & Target.Address(False, False) & _
        " aus Blatt " & ActiveSheet.Name & _
        " aus Arbeitsmappe " & ActiveWorkbook.Name & _
        " wurde geändert!"
End Sub
```

In der Workbook_Open-Prozedur wird die neue App-Klasse deklariert und initialisiert:

```
Dim AppClass As New clsApp
Private Sub Workbook_Open()
```

```
Set AppModule.App = Application
End Sub
```

14.3. Eine Ereignisprozedur für mehrere CommandButtons

In das Klassenmodul 'clsButton' wird zum einen eine Public-Variable für das Ereignis des CommandButton-Objekts und zum anderen der zugehörige Ereigniscode eingetragen:

```
Public WithEvents Btn As CommandButton

Private Sub Btn_Click()
    MsgBox "Aufruf erfolgt von Schaltfläche " & Right (Btn.Caption, 1)
End Sub
```

Die Deklaration und Initialisierung der Btn-Klasse erfolgt in der Workbook_Open-Prozedur (das Workbook muss übrigens ein Worksheet 'Buttons' mit

(mindestens) vier aus der Steuerelement-Toolbox eingefügten Befehlsschaltflächen beinhalten):

```
Dim CntBtn(1 To 4) As New clsButton

Private Sub Workbook_Open()
    Dim intCounter As Integer
    For intCounter = 1 To 4
        Set CntBtn(intCounter).Btn =
            ThisWorkbook.Worksheets("Buttons").OLEObjects(intCounter).Object
    Next intCounter
End Sub
```

14.4. Ein- und Auslesen einer Kundenliste

Zusätzlich zu diesen vordefinierten können neue, benutzerdefinierte Klassen geschaffen werden, mit denen es auf programmiertechnisch elegante Art möglich ist, eigene Typen zu bilden und z.B. mit Plausibilitätsprüfungsroutinen auf diese zuzugreifen.

In das Klassenmodul werden zum einen die Public-Variablen für Elemente des Kunden-Objekts und zum anderen eine Prüfroutine eingetragen:

```
Option Explicit
Public strNA As String
Public strNB As String
Public strS As String
Public strC As String
Public strPLZ As String

Property Let strP(strP As String)
    If Not IsNumeric(strP) Then
        MsgBox strP & " ist eine ungültige Postleitzahl"
        strPLZ = "?????"
    Else
        strPLZ = strP
    End If
End Property
```

Die Deklaration und die allgemeinen Codes werden in einem Standardmodul hinterlegt:

```

Dim NeuerKunde As New clsKunden
Dim colKunden As New Collection

Sub Einlesen ()
    Dim intCounter As Integer
    Set colKunden = Nothing
    For intCounter = 2 To 11
        Set NeuerKunde = New clsKunden
        With NeuerKunde
            .strNA = Cells(intCounter, 1).Value
            .strNB = Cells(intCounter, 2).Value
            .strS = Cells(intCounter, 3).Value
            .strP = Cells(intCounter, 4).Value
            .strC = Cells(intCounter, 5).Value
        End With
        colKunden.Add NeuerKunde
    Next intCounter
End Sub

Sub AdressenAusgeben ()
    Dim knd As clsKunden
    For Each knd In colKunden
        With knd
            MsgBox .strNA & vbLf & .strNB & vbLf & .strS & _

```

```
vbLf & .strPLZ & " " & .strC
End With
Next
End Sub
```

14.5. Ereignissteuerung einer Serie von Labels

Mit den nachfolgenden Prozeduren werden 256 Labels einer UserForm mit MouseMove, MouseClick- und anderen Ereignissen versehen.

In das Klassenmodul werden zum einen die Public-Variable für die Ereignisse des Label-Objekts und zum anderen die zugehörigen Ereigniscodes eingetragen:

```
Public WithEvents LabelGroup As MSForms.Label
Private Sub LabelGroup_Click ()
    With frmChar.txtString
        .Text = .Text & Me.LabelGroup.Caption
```

```

End With
End Sub

Private Sub LabelGroup_Db1Click( _
ByVal Cancel As MSForms.ReturnBoolean)
frmChar.txtString.Text = Me.LabelGroup.Caption
End Sub

Private Sub LabelGroup_MouseDown(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
Me.LabelGroup.ForeColor = &H80000009
Me.LabelGroup.BackColor = &H80000012
End Sub

Private Sub LabelGroup_MouseMove(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
Dim strChar As String
Dim intChar As Integer
frmChar.lblChar.Caption = Me.LabelGroup.Caption
strChar = Me.LabelGroup.Name
intChar = Cint(Right(strChar, Len(strChar) - 5)) - 1
frmChar.lblShortCut.Caption = "Alt+" & intChar
frmChar.lblZeichen.Caption = "=ZEICHEN(" & intChar & ")"
End Sub

```

```
Private Sub LabelGroup_MouseUp(ByVal Button As Integer, _  
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)  
    Me.LabelGroup.ForeColor = &H80000012  
    Me.LabelGroup.BackColor = &H80000009  
End Sub
```

Die Deklaration und Initialisierung der Labels-Klasse erfolgt in einem Standardmodul:

```
Dim Labels(1 To 256) As New clsFrm  
  
Sub clsSymbolAufruf()  
    Dim intCounter As Integer  
    For intCounter = 1 To 256  
        Set Labels(intCounter).LabelGroup = frmChar.Controls("Label" &  
intCounter)  
    Next intCounter  
    frmChar.Show  
End Sub
```


Teil IV.

**Weitergehende
Programmierkonzepte**

15. Code-Optimierung

Die folgende Grundsätze verhelfen zu einer optimalen Ablaufgeschwindigkeit Ihres VBA-Programms:

15.1. Konstanten

Deklarieren Sie, wo immer möglich, Konstanten statt Variablen.

15.2. Objektindex

Wenn es die Klarheit des Codes nicht stört, verwenden Sie bei Objekt-Schleifen den Index des Objektes, nicht den Namen.

```
Worksheets(intCounter)
```

ist schneller als

```
Worksheets("Tabelle1")
```

Allerdings gehen For-Each-Schleifen vor, denn

For Each wksData In Worksheets:**Next**

ist schneller als

```
Worksheets(intCounter)
```

15.3. Direkte Objektzuweisungen

Verwenden Sie keine allgemeinen Objektzuweisungen wie:

```
Dim wksData As Object
```

Deklarieren Sie korrekt:

```
Dim wksData As Worksheet
```

Dies hat auch den Vorteil, dass IntelliSense nach Eingabe eines Punktes Vorschläge machen kann, welche Eigenschaften und Methoden zu dem Objekt passen. Wenn die Objekte einer anderen Anwendung entstammen (z.B. Word oder Access), muss zunächst der Verweis auf die Objektbibliothek eingefügt werden, damit IntelliSense funktioniert.

15.4. Selektieren

Wählen Sie keine Arbeitsmappen, Blätter, Bereiche oder andere Objekte aus:

```
Workbooks("Test.xls").Activate  
Worksheets("Tabelle1").Select  
Range("A1").Select  
ActiveCell.Value = 12
```

Referenzieren Sie stattdessen exakt:

```
Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1").Value = 12
```

15.5. Keine eckigen Klammern

Verwenden Sie für Zellbereiche nicht die Schreibweise in eckigen Klammern:

```
[b3] = [d4]
```

Schreiben Sie stattdessen (Ausführungszeit ca. 66% von vorigem):

```
Range("B3").Value = Range("D4").Value
```

Noch etwas schneller (Ausführungszeit ca. 90% von vorigem bzw. 60% von ersterem):

```
Cells(3,2).Value = Cells(4,4).Value ' Cells (ZeilenNr, SpaltenNr)
```

Hinweis: Beachten Sie, dass bei Angabe des Zellbezug als String die Range-Eigenschaft verwendet werden muss, wohingegen bei der Angabe als Zahlen die Cells-Eigenschaft verwendet werden muss.

15.6. Direkte Referenzierung

Referenzieren Sie - wenn der Programmablauf es nicht erforderlich macht - nicht hierarchieweise:

```
Set wkbData = Workbooks("Test.xls")  
Set wksData = wkbData.Worksheets("Tabelle1")  
Set rngData = wksData.Range("A1:F16")
```

Referenzieren Sie stattdessen direkt das Zielobjekt:

```
Set rngData =  
Workbooks("Test.xls").Worksheets("Tabelle1").Range("A1:F16")
```

15.7. Dimensionierung

Dimensionieren Sie die Variablen nicht größer als dies erforderlich ist:

```
Dim intCounter As Integer  
ist schneller als:  
Dim varCounter as Variant
```

Vorsicht: Wenn eigentlich der Datentyp Byte ausreichen sollte, kann eine Subtraktion manchmal einen Unterlauf verursachen. Die Gefahr besteht vor allem bei FOR-Schleifen mit einem negativen Argument für STEP. In diesem Falle bei INTEGER bleiben.

Tipp: Noch etwas schneller als der Integer ist der Datentyp Long! Das liegt vermutlich daran, dass Integer 16-bittig ist während Long 32-bittig ist und alle neueren Prozessoren für 32-Bit optimiert sind.

15.8. With-Rahmen

Verwenden Sie With-Rahmen. Langsam ist:

```
Worksheets("Tabelle1").Range("A1:A16").Font.Bold = True  
Worksheets("Tabelle1").Range("A1:A16").Font.Size = 12  
Worksheets("Tabelle1").Range("A1:A16").Font.Name = "Arial"  
Worksheets("Tabelle1").Range("A1:A16").Value = "Hallo!"
```

Schneller ist:

```
With Worksheets("Tabelle1").Range("A1:A16")
With .Font
    .Bold = True
    .Size = 12
    .Name = "Arial"
End With
.Value = "Hallo!"
End With
```

15.9. Excel-Funktionen

Ziehen Sie Excel-Funktionen VBA-Routinen vor. Langsam ist:

```
For intCounter = 1 To 20
    dblSum = dblSum + Cells(intCounter, 1).Value
Next intCounter
```


Schneller ist:

```
dblSum = WorksheetFunction.Sum(Range("A1:A20"))
```

Wenn Sie große, zusammenhängende Zellbereich berechnen müssen, setzen Sie zur eigentlichen Berechnung Excel-Formeln ein. Die Formeln können Sie danach in absolute Werte umwandeln:

```
Sub Berechnen()  
    Dim intRow As Integer  
    intRow = Cells(Rows.Count, 1).End(xlUp).Row  
    Range("C1").Formula = "=A1+B1/Pi"  
    Range("C1:C" & intRow).FillDown  
    Columns("C").Copy  
    Columns("C").PasteSpecial Paste:=xlValues  
    Application.CutCopyMode = False  
    Range("A1").Select  
End Sub
```

Dasselbe Ergebnis hat folgende Prozedur, die auch With-Klammern verwendet und bei der Ersetzung der Formeln durch Werte ohne Copy/PasteSpecial auskommt:

```

Sub Berechnen2()
    Dim lngRow As Long
    lngRow = Cells(Rows.Count, 1).End(xlUp).Row
    With Range("C1:C" & lngRow)
        .Formula = "=A1+B1/Pi()" ' trägt die Formeln ein
        .Formula = .Value       ' ersetzt die Formeln durch Werte;
    .Value = .Value geht auch
    End With
    Range("A1").Select ' nur, wenn das nötig/erwünscht ist
End Sub

```

Tipp: Wenn Sie auf eine große Anzahl Zellen zugreifen müssen, dann ist es am Schnellsten, wenn Sie die Werte mit einem Befehl in ein Array kopieren und dann aus dem Array lesen:

```

Sub Berechne3()
    dim a
    dim i as long, j as long, sum as long
    a = me.Range("A1:H800").value
    for i=1 to 8
        for j=1 to 800
            sum=sum+a(j,i) ' a(ZeilenNr, SpaltenNr)

```

```
next j
next i
debug.print sum
End Sub
```

15.10. Array-Formeln

Setzen Sie temporäre Excel-Array-Formeln zur Matrixberechnung ein. Wenn Sie in VBA zwei Zellbereiche auf Übereinstimmung überprüfen wollen, müssen Sie einzelne Zellvergleiche vornehmen. Mit Einsatz einer Excel-Array-Formel sind Sie schneller. Im nachfolgenden Code werden zwei große Zellbereiche auf Übereinstimmung überprüft. Über VBA müsste man jede einzelne Zelle des einen mit der des anderen Bereiches vergleichen. Die Excel-Array-Formel liefert das Ergebnis unmittelbar nach dem Aufruf:

```
Function MatrixVergleich(strA As String, strB As String) As Boolean
    Range("IV1").FormulaArray = "=SUM((" & strA & "=" & strB & ") * 1)"
If Range("IV1").Value - Range(strA).Cells.Count = 0 Then
```

```
MatrixVergleich = True  
End If  
Range("IV1").ClearContents  
End Function  
Sub Aufruf()  
MsgBox MatrixVergleich("C1:D15662", "E1:F15662")  
End Sub
```

Teil V.

**Programmierbeispiele und
Prozedurvorlagen**

16. Menü- und Symbolleisten

16.1. Grundsätzliches

Menü- und Symbolleisten sind sowohl manuell wie auch über VBA zu erstellen, zu verändern und zu löschen.

Seit der Excel-Version 8.0 (Office 97) handelt es sich bei den Menü- und Symbolleisten um das Objektmodell der **Commandbars** mit den zugehörigen **Control**-Elementen *CommandBarButton*, *CommandBarPopUp* und *Command-*

dBarComboBox unter dem Oberbegriff *CommandBarControl*.

Grundsätzlich empfiehlt es sich, zu einer Arbeitsmappe gehörende *CommandBars* oder *CommandBarControls* beim Öffnen der Arbeitsmappe über das **Workbook_Open**-Ereignis zu erstellen und über das **Workbook_BeforeClose**-Ereignis zu löschen. Nur so ist gewährleistet, dass der Anwender nicht durch Auswirkungen von *CommandBar*-Programmierungen oder -Anbindungen belästigt wird.

Der *CommandBars*-Auflistung fügt man mit der **Add**-Methode eine neue Leiste hinzu. Erfolgt die Erstellung der neuen *CommandBar* in einem Klassenmodul, ist die Syntax **Application.CommandBars.Add...** zwingend erforderlich, erfolgt die Erstellung in einem Standardmodul, reicht ein **CommandBars.Add...** Um später mögliche Kollisionen mit anderen Office-Anwendungen zu vermeiden, wird allerdings auch hier die **Application**-Nennung empfohlen.

Die Add-Methode kann mit bis zu 9 Parameter aufgerufen werden:

- **Name**

Der Name der Symbolleiste, zwingend erforderlich

- **Position**

optional, folgende Konstanten sind möglich:

- `msoBarLeft` (am linken Bildschirmrand)
- `msoBarRight` (am rechten Bildschirmrand)
- `msoBarTop` (wird an die bestehenden Symbolleisten angegliedert)
- `msoBarBottom` (am unteren Bildschirmrand, über der Statusleiste)
- `msoBarFloating` (nicht verankerte Symbolleiste, die Position kann festgelegt werden)
- `msoBarPopUp` (Kontext-Symbolleiste, mit der rechten Maustaste im Tabellenblatt aufrufbar)

- **MenuBar**

optional, legt fest, ob es sich um eine Menü- oder eine Symbolleiste handelt (TRUE = Menüleiste, FALSE = Symbolleiste, Voreinstellung ist FALSE).

- **Temporary**

optional, legt fest, ob die Menü- oder Symbolleiste mit Microsoft Excel geschlossen werden soll (TRUE = temporär, FALSE = bestehenbleibend, Voreinstellung ist FALSE). Wird also TRUE festgelegt, wird die CommandBar gelöscht, wenn Excel geschlossen wird und taucht auch in der CommandBar-Auflistung nicht mehr auf.

16.2. Beispiele für das VBA-Handling von CommandBars

16.2.1. Menüleiste ein-/ausblenden

- Prozedur: CmdBarEinAus
- Art: Sub
- Modul: Standardmodul
- Zweck: Arbeitsblattmenüleiste aus- und einblenden.
- Ablaufbeschreibung:

- Rahmen mit dem CommandBar-Objekt bilden
- Wenn eingeschaltet ausschalten, sonst einschalten
- Code:

```

Sub CmdBarEinAus ()
    With Application.CommandBars ("Worksheet Menu Bar")
        .Enabled = Not .Enabled
    End With
End Sub

```

16.2.2. Neue Menüleiste erstellen und einblenden

- Prozedur: NewMenueBar
- Art: Sub
- Modul: Standardmodul
- Zweck: Es wird eine neue Menüleiste erstellt und eingeblendet, wobei die Arbeitsblattmenüleiste ausgeblendet wird.
- Ablaufbeschreibung:
 - Variablendeklaration
 - Prozedur zum Löschen der evtl. bereits bestehenden Menüleiste aufrufen

- Menüleiste erstellen
- 1. Menü erstellen
- Schleife über 12 Monate bilden
- Monatsschaltfläche erstellen
- Rahmen um das Schaltflächenobjekt erstellen
- Aufschrift festlegen
- Der Schaltfläche keine Prozedur zuweisen
- Den Aufschrifttyp festlegen
- 2. Menü erstellen
- Schleife über 12 Monate bilden
- Monatsschaltfläche erstellen
- Rahmen um das Schaltflächenobjekt erstellen
- Aufschrift festlegen
- Der Schaltfläche keine Prozedur zuweisen
- Den Aufschrifttyp festlegen
- Arbeitsblattmenüleiste ausblenden
- Neue Menüleiste einblenden
- Code:

```

Sub NewMenueBar ()
    Dim oCmdBar As CommandBar
    Dim oPopUp As CommandBarPopup
    Dim oCmdBtn As CommandBarButton
    Dim datDay As Date
    Dim iMonths As Integer
    Call DeleteNewMenueBar
    Set oCmdBar = Application.CommandBars.Add ( _
        Name:="MyNewCommandBar", _
        Position:=msoBarTop, _
        MenuBar:=True, _
        temporary:=True)
    Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
    oPopUp.Caption = "Prüfung"
    For iMonths = 1 To 12
        Set oCmdBtn = oPopUp.Controls.Add
        With oCmdBtn
            .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & "
Druck"
            .OnAction = ""
            .Style = msoButtonCaption
        End With
    Next iMonths
    Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)

```

```

oPopUp.Caption = "Monatsbericht"
For iMonths = 1 To 12
    Set oCmdBtn = oPopUp.Controls.Add
        With oCmdBtn
            .Caption = Format(DateSerial(1, iMonths, 1), "mmmm") & "
                Druck"
            .OnAction = ""
            .Style = msoButtonCaption
        End With
    Next iMonths
Application.CommandBars("Worksheet Menu Bar").Enabled = False
oCmdBar.Visible = True
End Sub

```

- **Prozedur: DeleteNewMenueBar**
- **Art: Sub**
- **Modul: Standardmodul**
- **Zweck: Evtl. bestehende Menüleiste löschen**
- **Ablaufbeschreibung:**
 - Fehleroutine für den Fall starten, dass die Menüleiste nicht existiert
 - Benutzerdefinierte Menüleiste löschen
 - Arbeitsblattmenüleiste einblenden

- Code:

```
Private Sub DeleteNewMenueBar ()  
    On Error GoTo ERRORHANDLER  
    Application.CommandBars ("MyNewCommandBar").Delete  
    Application.CommandBars ("Worksheet Menu Bar").Enabled = True  
Exit Sub  
ERRORHANDLER:  
End Sub
```

16.2.3. Alle Menüleisten ein-/ausblenden

- Prozedur: AllesAusEinBlenden
- Art: Sub
- Modul: Standardmodul
- Zweck: Alle Menü- und Symboleisten aus- und einblenden.
- Ablaufbeschreibung:
 - Objektvariable für CommandBar erstellen
 - Rahmen um das CommandBar-Objekt erstellen
 - Wenn die Arbeitsblattmenüleiste eingeblendet ist...

- Arbeitsblattmenüleiste ausblenden
- Auf Vollbildschirm schalten
- Eine Schleife über die CommandBars bilden
- Wenn es sich bei der aktuellen CommandBar nicht um die Arbeitsblattmenüleiste handelt...
- Wenn die aktuelle CommandBar sichtbar ist...
- Die aktuelle CommandBar ausblenden
- Aktive Arbeitsmappe schützen, wobei der Windows-Parameter auf **True** gesetzt wird (hierdurch werden die Anwendungs- und Arbeitsmappen-Schließkreuze ausgeblendet)
- Wenn die Arbeitsblattmenüleiste nicht sichtbar ist...
- Arbeitsmappenschutz aufheben
- Arbeitsblattmenüleiste anzeigen
- Vollbildmodus ausschalten
- Code:

```

Sub AllesAusEinBlenden ()
    Dim oBar As CommandBar
    With CommandBars ("Worksheet Menu Bar")

```



```

If .Enabled Then
    .Enabled = False
    Application.DisplayFullScreen = True
For Each oBar In Application.CommandBars
    If oBar.Name <> "Worksheet Menu Bar" Then
        If oBar.Visible Then
            oBar.Visible = False
        End If
    End If
Next oBar
ActiveWorkbook.Protect Windows:=True
Else
    ActiveWorkbook.Unprotect
    .Enabled = True
    Application.DisplayFullScreen = False
End If
End With
End Sub

```

16.2.4. Jahreskalender als Symbolleiste erstellen bzw. löschen

- Prozedur: NewCalendar
- Art: Sub

- Modul: Standardmodul
- Zweck: Jahreskalender als Symbolleiste anlegen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Fehleroutine einschalten
 - Jahreskalender-Symbolleiste löschen
 - Prozedur beenden
 - Wenn keine Jahreskalender-Symbolleiste vorhanden war...
 - Neue Symbolleiste erstellen
 - Schleife über 12 Monate bilden
 - Menü für jeden Monat anlegen
 - Menüaufschrift festlegen
 - Wenn der Monatszähler durch 4 teilbar ist, eine neue Gruppe beginnen
 - Die Tagesanzahl des jeweiligen Monats ermitteln
 - Eine Schleife über die Tage des jeweiligen Monats bilden
 - Das jeweilig aktuelle Datum ermitteln
 - Tagesschaltfläche erstellen
 - Aufschrift der Tagesschaltfläche festlegen

- Aufschriftart der Tagesschaltfläche festlegen
- Aufzurufende Prozedur festlegen
- Wenn es sich um einen Montag handelt, eine neue Gruppe beginnen
- Neue Symbolleiste anzeigen
- Code:

```

Sub NewCalendar ()
    Dim oCmdBar As CommandBar
    Dim oPopUp As CommandBarPopup
    Dim oCmdBtn As CommandBarButton
    Dim datDay As Date
    Dim iMonths As Integer, iDays As Integer, iCount As Integer
    On Error GoTo ERRORHANDLER
    Application.CommandBars (CStr(Year(Date))).Delete
    Exit Sub
ERRORHANDLER:
    Set oCmdBar = Application.CommandBars.Add( _
        CStr(Year(Date)), msoBarTop, False, True)
    For iMonths = 1 To 12
        Set oPopUp = oCmdBar.Controls.Add(msoControlPopup)
        With oPopUp
            .Caption = Format(DateSerial(1, iMonths, 1), "mmmm")
            If iMonths Mod 3 = 1 And iMonths <> 1 Then .BeginGroup =

```

```

True
    iCount = Day(DateSerial(Year(Date), iMonths + 1, 0))
For iDays = 1 To iCount
    datDay = DateSerial(Year(Date), iMonths, iDays)
    Set oCmdBtn = oPopUp.Controls.Add
    With oCmdBtn
        .Caption = Day(datDay) & " - " & Format(datDay,
            "dddd")
        .Style = msoButtonCaption
        .OnAction = "GetDate"
    If Weekday(datDay, vbUseSystemDayOfWeek) = 1 And iDays
    <> 1 Then .BeginGroup = True
        End With
    Next iDays
    End With
Next iMonths
    oCmdBar.Visible = True
End Sub

```

- Prozedur: GetDate
- Art: Sub
- Modul: Standardmodul
- Zweck: Das aufgerufene Tagesdatum melden
- Ablaufbeschreibung:

- Variablendeklaration
- Aktuelles Jahr ermitteln
- Monat ermitteln, aus dem der Aufruf erfolgte
- Tag ermitteln, der ausgewählt wurde
- Ausgewähltes Datum melden
- Code:

```

Sub GetData()
    Dim iYear As Integer, iMonth As Integer, iDay As Integer
    Dim iGroupM As Integer, iGroupD As Integer
    iYear = Year(Date)
    iMonth = WorksheetFunction.RoundUp(Application.Caller(2) - _
        (Application.Caller(2) / 4), 0)
    iDay = Application.Caller(1) - GetGroups(iMonth,
        Application.Caller(1))
    MsgBox Format(DateSerial(iYear, iMonth, iDay), "dddd - dd. mmmmm
        YYYY")
End Sub

```

- Prozedur: GetGroups
- Art: Function
- Modul: Standardmodul

- Zweck: Gruppe auslesen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Zählvariable initialisieren
 - Eine Schleife über alle Monate der Jahreskalender-Symbolleiste bilden
 - Solange die Zählvariable kleiner/gleich die Anzahl der Controls...
 - Wenn eine neue Gruppe beginnt...
 - Gruppenzähler um 1 hochzählen
 - Wenn die Zählvariable gleich dem übergebenen Tag minus dem Gruppenzähler, dann Schleife beenden
 - Zählvariable um 1 hochzählen
 - Gruppenzähler als Funktionswert übergeben
- Code:

```

Private Function GetGroups(iActMonth As Integer, iActDay As Integer)
  Dim iGroups As Integer, iCounter As Integer
  iCounter = 1
  With Application.CommandBars(CStr(Year(Date))).Controls(iActMonth)
    Do While iCounter <= .Controls.Count
      If .Controls(iCounter).BeginGroup = True Then

```

```
iGroups = iGroups + 1
End If
If iCounter = iActDay - iGroups Then Exit Do
iCounter = iCounter + 1
Loop
GetGroups = iGroups
End With
End Function
```

16.2.5. Alle Menü- und Symbolleisten auflisten

- Prozedur: ListAllCommandbars
- Art: Sub
- Modul: Standardmodul
- Zweck: Alle Symbolleisten mit dem englischen und dem Landesnamen mit der Angabe, ob sichtbar oder nicht, auflisten
- Ablaufbeschreibung:
 - Variablendeklaration
 - Bildschirmtaktualisierung ausschalten
 - Neue Arbeitsmappe anlegen

- Kopfzeile schreiben
- Kopfzeile formatieren
- Zeilenzähler initialisieren
- Eine Schleife über alle - eingebauten und benutzerdefinierten - CommandBars bilden
- Den englischen Namen eintragen
- Den Landesnamen eintragen
- Den Sichtbarkeitsstatus eintragen
- Spaltenbreiten automatisch anpassen
- Nicht genutzte Spalten ausblenden
- Nicht genutzte Zeilen ausblenden
- Bildschirmaktualisierung einschalten
- Speichernstatus der Arbeitsmappe auf WAHR setzen (um beim Schließen eine Speichern-Rückfrage zu übergehen)
- Code:

```

Sub ListAllCommandBars ()
    Dim oBar As CommandBar
    Dim iRow As Integer

```



```

Application.ScreenUpdating = False
Workbooks.Add 1
Cells(1, 1) = "Name"
Cells(1, 2) = "Lokaler Name"
Cells(1, 3) = "Sichtbar"
With Range("A1:C1")
    .Font.Bold = True
    .Font.ColorIndex = 2
    .Interior.ColorIndex = 1
End With
iRow = 1
For Each oBar In Application.CommandBars
    iRow = iRow + 1
    Cells(iRow, 1) = oBar.Name
    Cells(iRow, 2) = oBar.NameLocal
    Cells(iRow, 3) = oBar.Visible
Next oBar
Columns("A:C").AutoFit
Columns("D:IV").Hidden = True
Rows(iRow + 1 & ":" & Rows.Count).Hidden = True
Application.ScreenUpdating = True
ActiveWorkbook.Saved = True

End Sub

```

16.2.6. Jahreskalender bei Blattwechsel anlegen bzw. löschen

- Prozedur: Worksheet_Activate
- Art: Ereignis
- Modul: Klassenmodul des Arbeitsblattes **Dummy**
- Zweck: Jahreskalender-Symbolleiste erstellen
- Ablaufbeschreibung:
 - Aufruf der Prozedur zur Erstellung bzw. dem Löschen des Kalenders
- Code:

```
Private Sub Worksheet_Activate()  
    Call NewCalendar  
End Sub
```

- Prozedur: Worksheet_Deactivate
- Art: Ereignis
- Modul: Klassenmodul des Arbeitsblattes **Dummy**
- Zweck: Jahreskalender-Symbolleiste erstellen
- Ablaufbeschreibung:
 - Aufruf der Prozedur zur Erstellung bzw. dem Löschen des Kalenders

- Code:

```
Private Sub Worksheet_Deactivate()  
    Call NewCalendar  
End Sub
```

16.2.7. Dateinamen der *.xlb-Datei ermitteln

Die Informationen über die CommandBars werden in einer **.xlb**-Datei mit je nach Excel-Version wechselndem Namen im Pfad der Anwenderbibliotheken im Excel-Verzeichnis abgelegt. Die nachfolgenden Routinen ermitteln den Namen und das Änderungs-Datum dieser Datei. Der Code ist nur ab XL9 (Office 2000) lauffähig, da die **Application.UserLibraryPath**-Eigenschaft bei der Vorgängerversion noch nicht implementiert war. Der folgende Code nutzt das `Scripting.FileSystemObject` aus der `Scripting`-Klasse und setzt deshalb einen Verweis auf die "Microsoft Scripting Runtime"-Library voraus. Der Verweis kann im Makroeditor unter *Extras* > *Verweise* gesetzt werden. Ohne diesen Verweis kompiliert das Programm mit einem Fehler.

- Prozedur: `GetXLBName`

- Art: Sub
- Modul: Standardmodul
- Zweck: Name der XLB-Datei melden
- Ablaufbeschreibung:
 - Variablendeklaration
 - Funktion zur Ermittlung des Dateinamens aufrufen
 - Wenn ein Leerstring zurückgegeben wurde...
 - Negativmeldung
 - Sonst...
 - Meldung des Dateinamens
- Code:

```

Sub GetXLBName ()
  Dim sFile As String
  sFile = FindFile(0)
  If sFile = "" Then
    MsgBox "Die *.xlb-Datei wurde nicht gefunden!"
  Else
    MsgBox "Name der *.xlb-Datei: " & vbCrLf & sFile
  
```

End If

End Sub

- Prozedur: FindFile
- Art: Sub
- Modul: Standardmodul
- Zweck: Name und Änderungsdatum der XLB-Datei ermitteln
- Ablaufbeschreibung:
 - Variablendeklaration
 - Excel-Version ermitteln
 - Wenn es sich um die Version 8.0 handelt...
 - Negativmeldung und Prozedurende
 - Ein Sripting.FileSystemObject erstellen
 - Den Ordner oberhalb des Anwenderbibliothekspfads ermitteln und um den Begriff **Excel** erweitern
 - Eine Schleife über alle Dateien des ermittelten Ordners bilden
 - Wenn die Datei die Suffix **.xlb** beinhaltet...
 - Wenn das Änderungsdatum nach dem zuletzt ermittelten Änderungsdatum liegt...

- Änderungsdatum der aktuellen Datei in eine Datums-Variable einlesen
- Dateinamen in String-Variable einlesen
- Dateiname und Änderungsdatum in eine Variant-Variable einlesen
- Die Variant-Variable an die Funktion übergeben
- Code:

```

Private Function FindFile() As Variant
    Dim FSO As Scripting.FileSystemObject
    Dim oFile As Scripting.File
    Dim oFolder As Scripting.Folder
    Dim arrFile As Variant
    Dim datFile As Date
    Dim sFile As String, sVersion As String
    sVersion = Left(Application.Version, 1)
    If sVersion = "8" Then
        Beep
        MsgBox "Nur ab Version 9.0 möglich!"
    End
    End If
    Set FSO = New Scripting.FileSystemObject
    Set oFolder =

```

FSO.GetFolder(FSO.GetParentFolderName(Application.UserLibraryPath) &

```

"\Excel")
For Each oFile In oFolder.Files
    If Right(oFile.Name, 4) = ".xlb" Then
        If datFile < oFile.DateLastAccessed Then
            datFile = oFile.DateLastAccessed
            sFile = oFile.Path
        End If
    End If
Next oFile
arrFile = Array(sFile, datFile)
FindFile = arrFile
End Function

```

16.2.8. Dateänderungsdatum der *.xlb-Datei ermitteln

- Prozedur: GetXLBDate
- Art: Sub
- Modul: Standardmodul
- Zweck: Dateänderungsdatum der XLB-Datei melden
- Ablaufbeschreibung:
 - Variablendeklaration

- Funktion zur Ermittlung des Dateidatums aufrufen
- Wenn ein Nullwert zurückgegeben wurde...
- Negativmeldung
- Sonst...
- Meldung des Dateänderungsdatums
- Code:

```

Sub GetXLBDate ()
    Dim datFile As Date
    datFile = FindFile(1)
    If datFile = 0 Then
        MsgBox "Die *.xlb-Datei wurde nicht gefunden!"
    Else
        MsgBox "Letztes Änderungsdatum der *.xlb-Datei: " & vbCrLf &
        datFile
    End If
End Sub

```


17. Leeren und Löschen von Zellen

17.1. Über Dateieigenschaften

Über VBA-Prozeduren können Dateieigenschaften gelesen und geschrieben werden. Voraussetzung hierfür ist, dass das jeweilige Dokument geöffnet ist.

17.2. Programmierbeispiele

17.2.1. Dateieigenschaften lesen

- Prozedur: ReadDocumentProperties
- Art: Sub
- Modul: Standardmodul
- Zweck: Dateieigenschaften in eine Tabelle einlesen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Datenbereich leeren
 - Fehlerroutine starten
 - Rahmen um die BuiltInDocumentProperties bilden
 - Schleife über alle Elemente bilden
 - Den Namen der Eigenschaft eintragen
 - Den Wert der Eigenschaft eintragen
 - Den Typ der Eigenschaft eintragen

- Wenn ein Fehler aufgetreten ist...
- Den Fehlerwert eintragen
- Fehler-Objekt zurücksetzen
- Rahmen um die CustomDocumentProperties bilden
- Schleife über alle Elemente bilden
- Den Namen der Eigenschaft eintragen
- Den Wert der Eigenschaft eintragen
- Den Typ der Eigenschaft eintragen
- Wenn ein Fehler aufgetreten ist...
- Den Fehlerwert eintragen
- Fehler-Objekt zurücksetzen
- Code:

```

Sub ReadDocumentProperties ()
    Dim iRow As Integer
    Range ("A4:F35").ClearContents
    On Error Resume Next
    With ActiveWorkbook.BuiltinDocumentProperties
        For iRow = 1 To .Count

```

```
Cells(iRow + 3, 1).Value = .Item(iRow).Name
Cells(iRow + 3, 2).Value = .Item(iRow).Value
Cells(iRow + 3, 3).Value = .Item(iRow).Type
If Err.Number <> 0 Then
    Cells(iRow + 3, 2).Value = CVErr(xlErrNA)
    Err.Clear
End If
Next iRow
End With
With ActiveWorkbook.CustomDocumentProperties
For iRow = 1 To .Count
    Cells(iRow + 3, 5).Value = .Item(iRow).Name
    Cells(iRow + 3, 6).Value = .Item(iRow).Value
    Cells(iRow + 3, 7).Value = .Item(iRow).Type
If Err.Number <> 0 Then
        Cells(iRow + 3, 6).Value = CVErr(xlErrNA)
        Err.Clear
End If
Next iRow
End With
On Error GoTo 0
End Sub
```

17.2.2. Dateieigenschaften schreiben

- Prozedur: WriteDocumentProperties
- Art: Sub
- Modul: Standardmodul
- Zweck: Dateieigenschaften in eine Datei schreiben
- Ablaufbeschreibung:
 - Variablendeklaration
 - Aktives Blatt an eine Objekt-Variable übergeben
 - Wenn die Zelle A4 leer ist...
 - Warnton
 - Warnmeldung
 - Prozedur verlassen
 - Neue Arbeitsmappe anlegen
 - Rahmen um die BuiltInDocumentProperties bilden
 - Eine Schleife um den Datenbereich bilden
 - Wenn die Zelle in Spalte A der aktuellen Zeile leer ist, Prozedur verlassen
 - Wenn sich in Spalte B der aktuellen Zeile kein Fehlerwert befindet...

- Wert für die Dateieigenschaft gem. Spalte A der aktuellen Zeile festlegen
- Rahmen um die CustomDocumentProperties bilden
- Eine Schleife um den Datenbereich bilden
- Eine benutzerdefinierte Eigenschaft hinzufügen
- Vollzugsmeldung anzeigen
- Code:

```

Sub WriteDocumentProperties()
    Dim wks As Worksheet
    Dim iRow As Integer
    Set wks = ActiveSheet
    If IsEmpty(Range("A4")) Then
        Beep
        MsgBox "Sie müssen zuerst die Eigenschaften einlesen!"
    Exit Sub
End If
    Worksheets.Add
    With ActiveWorkbook.BuiltinDocumentProperties
        For iRow = 4 To 35
            If IsEmpty(wks.Cells(iRow, 1)) Then Exit For
            If IsError(wks.Cells(iRow, 2)) = False Then
                .Item(wks.Cells(iRow, 1).Value) = wks.Cells(iRow,

```

```

2) .Value
    End If
    Next iRow
End With
With ActiveWorkbook.CustomDocumentProperties
    For iRow = 4 To 4
        .Add Name:=wks.Cells(iRow, 5).Value, LinkToContent:=False, _
            Type:=msoPropertyTypeDate, Value:=wks.Cells(iRow,
6) .Value
    Next iRow
End With
MsgBox "Die editierbaren Dateieigenschaften wurden auf diese neue"
& vbCrLf & _
    "Arbeitsmappe übertragen, bitte prüfen."
End Sub

```


18. Leeren und Löschen von Zellen

18.1. Löschen aller leeren Zellen einer Spalte

```
Sub DeleteEmptyCells()  
    Dim intLastRow As Integer  
    Dim intRow As Integer  
    intLastRow = Cells.SpecialCells(xlCellTypeLastCell).Row  
    For intRow = intLastRow To 1 Step -1  
        If Application.CountA(Rows(intRow)) = 0 Then  
            intLastRow = intLastRow - 1  
        Else  
            Exit For  
        End If  
    End If
```

```

Next intRow
For intRow = intLastRow To 1 Step -1
    If IsEmpty(Cells(intRow, 1)) Then
        Cells(intRow, 1).Delete xlShiftUp
    End If
Next intRow
End Sub

```

18.2. Löschen der Zeile, wenn Zelle in Spalte A leer ist

```

Sub DeleteRowIfEmptyCell()
    Dim intRow As Integer, intLastRow As Integer
    intLastRow = Cells.SpecialCells(xlCellTypeLastCell).Row
    For intRow = intLastRow To 1 Step -1
        If Application.CountA(Rows(intRow)) = 0 Then
            intLastRow = intLastRow - 1
        Else
            Exit For
        End If
    Next intRow
    For intRow = intLastRow To 1 Step -1

```

```

If IsEmpty(Cells(intRow, 1)) Then
    Rows(intRow).Delete
End If
Next intRow
End Sub

```

18.3. Löschen aller leeren Zeilen

```

Sub DeleteEmptyRows ()
    Dim intRow As Integer, intLastRow As Integer
    intLastRow = Cells.SpecialCells(xlCellTypeLastCell).Row
    For intRow = intLastRow To 1 Step -1
        If Application.CountA(Rows(intRow)) = 0 Then
            Rows(intRow).Delete
        End If
    Next intRow
End Sub

```

18.4. FehlerZellen leeren

```
SubClearContentsErrorCells()  
  On Error GoTo ERRORHANDLER  
  Cells.SpecialCells(xlCellTypeFormulas, 16).ClearContents  
ERRORHANDLER:  
End Sub
```

18.5. FehlerZellen löschen

```
SubClearErrorCells()  
  On Error GoTo ERRORHANDLER  
  Cells.SpecialCells(xlCellTypeFormulas, 16).Delete xlShiftUp  
ERRORHANDLER:  
End Sub
```

18.6. Löschen aller Zellen in Spalte A mit "hallo" im Text

```
Sub DeleteQueryCells()  
    Dim var As Variant  
    Do While Not IsError(var)  
        var = Application.Match("hallo", Columns(1), 0)  
        If Not IsError(var) Then Cells(var, 1).Delete xlShiftUp  
    Loop  
End Sub
```

18.7. Leeren aller Zellen mit gelbem Hintergrund

```
Sub ClearYellowCells()  
    Dim rng As Range  
    For Each rng In ActiveSheet.UsedRange  
        If rng.Interior.ColorIndex = 6 Then  
            rng.ClearContents  
        End If  
    Next rng  
End Sub
```

```
End If
Next rng
End Sub
```

18.8. Alle leeren Zellen löschen

```
Sub DeleteEmptyys ()
    Dim rng As Range
    Application.ScreenUpdating = False
    For Each rng In ActiveSheet.UsedRange
        If IsEmpty(rng) Then rng.Delete xlShiftUp
    Next rng
    Application.ScreenUpdating = True
End Sub
```

19. XL4-Makros in VBA verwenden

19.1. Zum Aufruf von XL4-Makros in VBA

Es gibt Bereiche – beispielsweise das Setzen oder Auslesen der PageSetup-Eigenschaften –, in denen VBA deutliche Performance-Nachteile gegenüber alten XL4-Makros aufzeigt. Zudem bieten XL4-Makros Features, die von den VBA-Entwicklern nicht mehr berücksichtigt wurden. Dazu gehört unter anderem die Möglichkeit, Werte aus geschlossenen Arbeitsmappen auszulesen. Der Aufruf von XL4-Makros ist – wie in den nachfolgenden Prozeduren

gezeigt wird – aus VBA heraus möglich. Man beachte die Laufzeitschnelligkeit im Vergleich zu VBA-Makros.

19.2. Programmierbeispiele

Tabelle FalseLinks

19.3. Auslesen eines Wertes aus geschlossener Arbeitsmappe

```
Function xl4Value(strParam As String) As Variant
    xl4Value = ExecuteExcel4Macro(strParam)
End Function

Sub CallValue()
    Dim strSource As String
```



```

strSource = _
    "" & _
    Range("A2").Text & _
    "\" & Range("B2").Text & _
    "]" & Range("C2").Text & _
    "!" & Range("D2").Text
MsgBox "Zellwert Zelle A1: " & xl4Value(strSource)
End Sub

```

oder:

```

Sub Zelle_auslesen()
    Dim Adresse As String, Zeile As Integer, Spalte As Integer,
    Zellbezug As String
    Pfad = "D:\neue Dokumente\"
    Datei = "Urlaub 2009.xls"
    Register = "Kalender"
    Zeile = 14: Spalte = 20 ' entspricht T14
    Zellbezug = Cells(Zeile, Spalte).Address(ReferenceStyle:=xlR1C1)

    Adresse = "" & Pfad & "[" & Datei & "]" & Register & "!" &
    Zellbezug

    Ergebnis = ExecuteExcel4Macro(Adresse)
    MsgBox ("Wert der Zelle T14: " & Ergebnis)

```

End Sub

19.4. Auslesen des ANZAHL2-Wertes aus geschlossener Arbeitsmappe

```
Function xl4CountA(strParam As String) As Variant
    xl4CountA = _
        ExecuteExcel4Macro("CountA(" & strParam & ")")
End Function

Sub CallCountA()
    Dim strSource As String
    strSource = _
        "'1' & _
        Range("A3").Text & _
        "\[" & Range("B3").Text & _
        "]" & Range("C3").Text & _
        "'!'" & Range("D3").Text
```

```
MsgBox "ANZAHL2 in A1:A100: " & xl4CountA(strSource)
End Sub
```

19.5. Auslesen einer Summe aus geschlossener Arbeitsmappe

```
Function xl4Sum(strParam As String) As Variant
    xl4Sum = _
        ExecuteExcel4Macro("Sum(" & strParam & ")")
End Function

Sub CallSum()
    Dim strSource As String
    strSource = _
        "" & _
        Range("A4").Text & _
        "\[" & Range("B4").Text & _
        "]" & Range("C4").Text & _
        "!" & Range("D4").Text
```

```
MsgBox "SUMME in A1:B100: " & xl4Sum(strSource)
End Sub
```

19.6. Auslesen eines SVERWEIS-Wertes aus geschlossener Arbeitsmappe

```
Function xl4VLookup(strParam As String) As Variant
xl4VLookup = ExecuteExcel4Macro _
    ("VLookup("" & Range("E5").Text & _
    """, "" & strParam & "", "" & _
    Range("F5").Text & "", "" & _
    Range("G5").Text & "")")
End Function
```

```
Sub CallVLookup()
Dim strSource As String
strSource = _
    "" & _
    Range("A5").Text & _
    "\" & Range("B5").Text & _
```

```

"]" & Range("C5").Text & _
"!)" & Range("D5").Text
MsgBox "SVERWEIS in A1:B100: " & _
xl4VLookup(strSource)
End Sub

```

19.7. Auslesen einer Tabelle aus geschlossener und Einlesen in neue Arbeitsmappe

```

Sub ReadTable ()
Dim wks As Worksheet
Dim intRow As Integer, intCol As Integer
Dim strSource As String
Application.ScreenUpdating = False
Set wks = ActiveSheet
Worksheets.Add
For intRow = 1 To 20
    For intCol = 1 To 2
        strSource = _
            " " & _

```

```

wks.Range("A3").Text & _
"\[" & wks.Range("B2").Text & _
"]" & wks.Range("C2").Text & _
"!R" & intRow & "C" & intCol
Cells(intRow, intCol).Value = _
xl4Value(strSource)
Next intCol
Next intRow
Application.ScreenUpdating = True
End Sub

```

19.8. SVERWEIS aus XL4 anwenden

Bei Eingabe eines Suchbegriffes in Spalte A SVERWEIS-Wert in Spalte B eintragen. Der Code muss sich im Klassenmodul der Tabelle befinden. Die Daten werden aus der geschlossenen Arbeitsmappe ohne Formeleinsatz ausgelesen.

```

Private Sub Worksheet_Change(ByVal Target As Range)
Dim strSource As String
If Target.Column <> 1 Then Exit Sub

```

```

With Worksheets("FalseLinks")
    strSource = _
        "" & _
        .Range("A5").Text & _
        "\[" & .Range("B5").Text & _
        "]" & .Range("C5").Text & _
        "!" & .Range("D5").Text
End With
Target.Offset(0, 1).Value = _
    xl4VLookupEvent(strSource, Target.Text)
End Sub

Private Function xl4VLookupEvent ( _
    strParam As String, _
    strFind As String) As Variant
    With Worksheets("FalseLinks")
        xl4VLookupEvent = _
            ExecuteExcel4Macro("VLookup(" & strFind & _
                """, " & strParam & ", " & _
                .Range("F5").Text & ", " & _
                .Range("G5").Text & ")")
    End With
End Function

```

19.9. Namen über XL4 erstellen und ausblenden

Über XL4-Makros können Namen vergeben werden, die über die VBA-Eigenschaft `Visible` nicht angezeigt und den Befehl `Delete` nicht gelöscht werden können. Die Namen sind in allen Arbeitsmappen gültig und können als globale Variablen benutzt werden. Ihre Lebensdauer ist abhängig von der Excel-Sitzung. Routine zum Erstellen, Aufrufen und Löschen einer Text-Konstanten:

```
Sub setHiddenConst()  
    Dim txt As String  
    txt = InputBox("Bitte beliebige Meldung eingeben:", , _  
        "Dies ist meine konstante Meldung!")  
    If txt = "" Then Exit Sub  
    Application.ExecuteExcel4Macro _  
        "SET.NAME ("MyMsg", "" & txt & "")"  
End Sub  
  
Sub GetHiddenConst()  
    On Error Resume Next  
    MsgBox Application.ExecuteExcel4Macro ("MyMsg")
```



```

If Err > 0 Then
    Beep
    Err.Clear
    MsgBox "Es wurde keine Konstante initialisiert!"
End If
On Error GoTo 0
End Sub

Sub DeleteHiddenConst ()
    Application.ExecuteExcel4Macro "SET.NAME("MyMsg")"
End Sub

```

19.10. Benannte Formel über XL4 anlegen und aufrufen

Routine zum Erstellen, Aufrufen und Löschen der Osterformel.

```

Sub SetHiddenEastern ()
    Application.ExecuteExcel4Macro _
        "SET.NAME("OSTERN", "=FLOOR(DATE(MyYear, 3, " & _

```

```

"MOD(18.37*MOD(MyYear,19)-6,29),7)+29" ) "
End Sub

Sub GetHiddenEastern()
On Error Resume Next
MsgBox Format(Evaluate( _
Application.ExecuteExcel4Macro("OSTERN")), _
"dd.mm.yyyy")
If Err > 0 Then
Beep
Err.Clear
MsgBox "Es wurde kein Ostern initialisiert!"
End If
On Error GoTo 0
End Sub

Sub DeleteHiddenEastern()
Application.ExecuteExcel4Macro "SET.NAME("OSTERN")"
End Sub

```

19.11. Routine zum Erstellen, Aufrufen und Löschen der Kalenderwochen-Formel

```
Sub SetHiddenKW ()
    Application.ExecuteExcel4Macro _
        "SET.NAME ("DINKW", "=TRUNC ((MYWK-WEEKDAY (MYWK, 2) -" & _
        "DATE (YEAR (MYWK+4-WEEKDAY (MYWK, 2)), 1, -10)) / 7) """)"
End Sub

Sub GetHiddenKW ()
    On Error Resume Next
    MsgBox Evaluate(Application.ExecuteExcel4Macro ("DINKW"))
    If Err > 0 Then
        Beep
        Err.Clear
        MsgBox "Es wurde keine Kalenderwoche initialisiert!"
    End If
    On Error GoTo 0
End Sub

Sub DeleteHiddenKW ()
```

```
Application.ExecuteExcel4Macro "SET.NAME("DINKW")"  
End Sub
```

19.12. Druckprogrammierung über XL4-Makros

Wesentliche Geschwindigkeitsvorteile werden erreicht, wenn XL4-Makros beim Auslesen oder beim Setzen von PageSetup-Eigenschaften eingesetzt werden.

Auslesen der Seitenzahl des aktiven Blattes

```
Sub PageCountActiveSheet()  
MsgBox "Seitenanzahl: " & _  
ExecuteExcel4Macro("GET.DOCUMENT(50)")  
End Sub
```

Auslesen der Seitenanzahl eines andere Blattes

```
Sub PageCountOtherSheet ()
MsgBox "Seitenanzahl: " & _
ExecuteExcel4Macro ("Get.document (50, ""DeleteRows"") ")
End Sub
```

Auslesen der Seitenanzahl eines Blattes in einer anderen Arbeitsmappe

```
Sub PageCountOtherWkb ()
Dim wkb As Workbook
On Error Resume Next
Set wkb = Workbooks ("Test.xls")
If Err > 0 Or wkb Is Nothing Then
Beep
MsgBox "Es muss eine Arbeitsmappe ""Test.xls"" geöffnet sein!"
Exit Sub
End If
MsgBox "Seitenanzahl: " & _
ExecuteExcel4Macro ("Get.document (50, ""[Test.xls]Tabelle1"") ")
End Sub
```

Setzen von Druckeigenschaften wie Schriftgröße, Schriftart u.ä.

```

Sub SetPageSetup()
ExecuteExcel4Macro _
    "PAGE.SETUP(" & L & "" & "Arial,Bold" & _
    "8MeineFirma GmbH & Co. KG&R&" & "Arial,Bold" & _
    "&D,Seite 1" & ",0.75,0.75,0.91,0.5,FALSE,FALSE,TRUE,FALSE" & _
    ",2,1,95,#N/A,1,TRUE,,0.75,0.25,FALSE,FALSE)"
End Sub

```

Auslesen aller horizontalen und vertikalen Seitenumbrüche

```

Sub GetPageBreaks()
Dim horzpbArray() As Integer
Dim vertpbArray() As Integer
Dim intCounter As Integer, intCol As Integer, intRow As Integer

ThisWorkbook.Names.Add Name:="hzPB", _
    RefersToR1C1:="=GET.DOCUMENT(64," & "PrintPages")
ThisWorkbook.Names.Add Name:="vPB", _
    RefersToR1C1:="=GET.DOCUMENT(65," & "PrintPages)"
intCounter = 1
While Not IsError(Evaluate("Index(hzPB," & intCounter & ")"))
    ReDim Preserve horzpbArray(1 To intCounter)
    horzpbArray(intCounter) = Evaluate("Index(hzPB," & intCounter &

```

```

")")
    intCounter = intCounter + 1
Wend
ReDim Preserve horzpbArray(1 To intCounter - 1)
intCounter = 1
While Not IsError(Evaluate("Index(vPB," & intCounter & ")"))
    ReDim Preserve verpbArray(1 To intCounter)
    verpbArray(intCounter) = Evaluate("Index(vPB," & intCounter &
")")
    intCounter = intCounter + 1
Wend
ReDim Preserve verpbArray(1 To intCounter - 1)
Workbooks.Add
With Range("A1")
    .Value = "Horizontale Seitenumbrüche (Zeilen):"
    .Font.Bold = True
End With
For intRow = LBound(horzpbArray, 1) To UBound(horzpbArray, 1)
    Cells(intRow + 1, 1) = horzpbArray(intRow)
Next intRow
With Range("B1")
    .Value = "Vertikale Seitenumbrüche (Spalten):"
    .Font.Bold = True
End With
For intCol = LBound(verpbArray, 1) To UBound(verpbArray, 1)

```

```

Cells(intCol + 1, 2) = verpbArray(intCol)
Next intCol
Columns.AutoFit
Columns("A:B").HorizontalAlignment = xlCenter
End Sub

```

19.13. Schließen der Arbeitsmappe verhindern

In den Excel-Versionen ab XL8 kann über das Workbook_BeforeClose-Ereignis das Schließen der Arbeitsmappe verhindert werden. Dieses Ereignis steht bei der Vorgängerversionen nicht zur Verfügung. Wenn also eine Arbeitsmappe abwärtskompatibel sein soll, kann hier ein XL4-Makro eingesetzt werden.

```

Sub auto_close()
If Worksheets("NoClose").CheckBoxes _
    ("chbClose").Value = xlOn Then
    ExecuteExcel4Macro "HALT(TRUE)"
    MsgBox "Das Schließen der Arbeitsmappe " & _

```



```
"ist gesperrt -" & vbLf & _  
"Bitte zuerst die Sperre im " & _  
"Blatt " & "NoClose" aufheben!"
```

```
End If  
End Sub
```

19.14. Arbeitsblattmenüleiste zurücksetzen

Über Schaltfläche kann die Arbeitsblattmenüleiste zurückgesetzt und die letzte Einstellung wieder gesetzt werden

```
Sub MenuBar()  
With ActiveSheet.Buttons(1)  
If .Caption = "Menüleiste Reset" Then  
ExecuteExcel4Macro "SHOW.BAR(2)"  
.Caption = "Menüleiste zurück"  
Else  
ExecuteExcel4Macro "SHOW.BAR(1)"  
.Caption = "Menüleiste Reset"  
End If
```

```
End With  
End Sub
```

19.15. Bedingtes Löschen von Zeilen

Das Löschen von Zeilen nach bestimmten Kriterien kann in VBA eine zeitaufwendige Aufgabe sein, mit XL4-Makros ist das vergleichsweise schnell und einfach zu lösen

```
Sub DeleteRows()  
    Dim rngAll As Range, rngCriteria As Range  
    Application.ScreenUpdating = False  
    Set rngAll = Range("A1").CurrentRegion  
    rngAll.Name = "" & ActiveSheet.Name & "!Datenbank"  
    Set rngCriteria = rngAll.Resize(2, 1).Offset _  
        (0, rngAll.Columns.Count + 1)  
    With rngCriteria  
        .Name = "" & ActiveSheet.Name & _  
            "!Suchkriterien"  
        .Cells(1, 1).Value = "Name"
```

```
.Cells(2, 1).Formula = "'<>Hans W. Herber"  
ExecuteExcel4Macro "DATA.DELETE ()"  
.Clear  
End With  
Application.ScreenUpdating = True  
End Sub
```


20. Textimport

20.1. Import zur Anzeige in MsgBoxes

Beim Import mit der Funktion Line Input sucht Excel nach Zeichen, die das Zeilenende ankündigen. Wurde eine Datei unter Windows geschrieben, endet eine Zeile üblicherweise mit zwei Zeichen: CHR(13) und CHR(10), also Wagenrücklauf (CR = Carriage Return) und Zeilenvorschub (LF = LineFeed). Mac-Dateien enden üblicherweise mit CHR(13) und Unix-Dateien enden üblicherweise mit CHR(10). 'Üblicherweise' meint, dass dies für Textdateien gilt, die das Betriebssystem schreibt und die als Konvention auch so von vielen

Anwendungen von ihrem jeweiligen Betriebssystem übernommen wird. Es gibt aber auch Anwendungen, die auf mehreren Betriebssystemen laufen und andere oder überall die gleiche Konvention für das Zeilenende verwenden.

Excel gibt es für Windows und Mac, daher werden von Line Input sowohl CR+LF als auch CR als Zeilenendzeichen erkannt. Ein einfaches LF oder andere Symbole werden versteht Excel nicht als Zeilenende und liest dann so lange ein, bis der Puffer voll ist – die eingelesene Zeichenfolge kann in diesem Falle mehrere zehntausend Byte lang werden.

```
Sub WriteInMsgBoxes ()
    Dim cIn As New Collection
    Dim arrAct As Variant
    Dim intNo As Integer, intCounter As Integer
    Dim txt As String, strMsg As String
    Dim bln As Boolean
    intNo = FreeFile
    Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intNo

    Do Until EOF(intNo)
        If bln = False Then
```

```
Line Input #intNo, txt
arrAct = SplitString(txt, ",")
For intCounter = 1 To UBound(arrAct)
    cln.Add arrAct(intCounter)
Next intCounter
Else
    Line Input #intNo, txt
    arrAct = SplitString(txt, ",")
    For intCounter = 1 To UBound(arrAct)
        strMsg = strMsg & cln(intCounter) & ": " & _
            arrAct(intCounter) & vbCrLf
    Next intCounter
End If
If bln Then MsgBox strMsg
bln = True
strMsg = ""
Loop
Close intNo
End Sub
```

20.2. Import zur Konvertierung in eine HTML-Seite

```
Sub WriteInHTML ()
    Dim arrAct As Variant
    Dim intSource, intTarget, intCounter As Integer
    Dim txt, strTag As String
    Dim bln As Boolean
    intTarget = FreeFile
    Open ThisWorkbook.Path & "\TextImport.htm" For Output As
#intTarget
    Print #intTarget, "<html><body><table>"
    intSource = FreeFile
    Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource

    Do Until EOF(intSource)
        If bln Then strTag = "td" Else strTag = "th"
        Line Input #intSource, txt
        arrAct = SplitString(txt, ",")
        Print #intTarget, "<tr>"
        For intCounter = 1 To UBound(arrAct)
            Print #intTarget, "<" & strTag & ">" & arrAct(intCounter) &
"</" & strTag & ">"
        Next intCounter
    End Do
End Sub
```



```

Print #intTarget, "</tr>"
  bln = True
Loop
Close intSource
Print #intTarget, "</table></body></html>"
Close intTarget
Shell "hh " & ThisWorkbook.Path & "\TextImport.htm",
vbMaximizedFocus
End Sub

```

20.3. Import zur Anzeige in einem Arbeitsblatt

```

Sub WriteInWks()
  Dim cln As New Collection
  Dim arrAct As Variant
  Dim intSource As Integer, intRow As Integer, intCol As Integer
  Dim txt As String
  Worksheets.Add
  intSource = FreeFile
  Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource

```

```

Do Until EOF(intSource)
  Line Input #intSource, txt
  arrAct = SplitString(txt, ",")
  intRow = intRow + 1
  For intCol = 1 To UBound(arrAct)
    Cells(intRow, intCol).Value = arrAct(intCol)
  Next intCol
Loop
Close intSource
Rows(1).Font.Bold = True
End Sub

```

20.4. Import zur Übernahme in UserForm-Controls

In einem Standardmodul:

```

Public garr() As String
Public gint As Integer

```

Im Klassenmodul der UserForm:

```

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdWeiter_Click()
    Dim intCounter As Integer
    If gint <= 4 Then gint = gint + 1 Else gint = 1
    For intCounter = 1 To 5
        Controls("TextBox" & intCounter).Text = garr(gint, intCounter)
    Next intCounter
End Sub

Private Sub UserForm_Initialize()
    Dim arrAct As Variant
    Dim intSource As Integer, intCounter As Integer, intRow As Integer

    Dim txt As String
    Dim bln As Boolean
    gint = 0
    intSource = FreeFile
    Open ThisWorkbook.Path & "\TextImport.txt" For Input As #intSource

    Do Until EOF(intSource)
        Line Input #intSource, txt

```

```

arrAct = SplitString(txt, ",")
If bln = False Then
    For intCounter = 1 To UBound(arrAct)
        Controls("Label" & intCounter).Caption = _
            arrAct(intCounter) & ":"
    Next intCounter
    ReDim garr(1 To 5, 1 To UBound(arrAct))
Else
    intRow = intRow + 1
    For intCounter = 1 To UBound(arrAct)
        garr(intRow, intCounter) = arrAct(intCounter)
    Next intCounter
End If
    bln = True
Loop
    Close intSource
End Sub

```

Für alle vorstehende Routinen wird die folgende benutzerdefinierte Funktion in einem Standardmodul benötigt (Die Funktion macht unabhängig von der erst ab XL2000 verfügbaren VBA-Funktion **Split**):

```

Function SplitString(ByVal txt As String, strSeparator As String)

```

```
Dim arr() As String
Dim intCounter As Integer
Do
    intCounter = intCounter + 1
    ReDim Preserve arr(1 To intCounter)
    If InStr(txt, strSeparator) Then
        arr(intCounter) = Left(txt, InStr(txt, strSeparator) - 1)
        txt = Right(txt, Len(txt) - InStr(txt, strSeparator))
    Else
        arr(intCounter) = txt
        Exit Do
    End If
Loop
SplitString = arr
End Function
```


21. Sortieren

Auf die folgenden 3 Codes greifen mehrere der Sortierprogramme zu:

21.1. Schnelle VBA-Sortierroutine

Autor: *John Green*

```
Sub QuickSort(ByRef VA_array, Optional V_Low1, Optional V_High1)
Dim V_Low2 As Long, V_High2 As Long
Dim V_val1 As Variant, V_val2 As Variant
```

```
If IsMissing(V_Low1) Then  
    V_Low1 = LBound(VA_array, 1)  
End If  
If IsMissing(V_high1) Then  
    V_High1 = UBound(VA_array, 1)  
End If  
V_Low2 = V_Low1  
V_High2 = V_High1  
V_val1 = VA_array((V_Low1 + V_High1) / 2)  
While (V_Low2 <= V_High2)  
    While (VA_array(V_Low2) < V_val1 And _  
        V_Low2 < V_High1)  
        V_Low2 = V_Low2 + 1  
    Wend  
    While (VA_array(V_High2) > V_val1 And _  
        V_High2 > V_Low1)  
        V_High2 = V_High2 - 1  
    Wend  
    If (V_Low2 <= V_High2) Then  
        V_val2 = VA_array(V_Low2)  
        VA_array(V_Low2) = VA_array(V_High2)  
        VA_array(V_High2) = V_val2  
        V_Low2 = V_Low2 + 1  
        V_High2 = V_High2 - 1  
    End If
```



```

Wend
If (V_High2 > V_Low1) Then Call _
    QuickSort(VA_array, V_Low1, V_High2)
If (V_Low2 < V_High1) Then Call _
    QuickSort(VA_array, V_Low2, V_High1)

End Sub

```

21.2. Dialog zur Verzeichnisauswahl

```

Public Type BROWSEINFO
    howner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

```

```

Declare Function SHGetPathFromIDLList Lib "shell32.dll" _

```

```

Alias "SHGetPathFromIDListA" (ByVal pidl As Long, _
ByVal pszPath As String) As Long

Declare Function SHBrowseForFolder Lib "shell32.dll" _
Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) As Long

Function GetDirectory(Optional msg) As String
    Dim bInfo As BROWSEINFO
    Dim Path As String
    Dim r As Long, x As Long, pos As Integer
    bInfo.pidlRoot = 0&
    If IsMissing(msg) Then
        bInfo.lpszTitle = "Wählen Sie bitte einen Ordner aus."
    Else
        bInfo.lpszTitle = msg
    End If
    bInfo.ulFlags = &H1
    x = SHBrowseForFolder(bInfo)
    Path = Space$(512)
    r = SHGetPathFromIDList(ByVal x, ByVal Path)
    If r Then
        pos = InStr(Path, Chr$(0))
        GetDirectory = Left(Path, pos - 1)
    Else

```

```
GetDirectory = ""  
End If  
End Function
```

21.3. Auslesen der Dateinamen in einem Verzeichnis

```
Function FileArray(strPath As String, strPattern As String)  
Dim arrDateien()  
Dim intCounter As Integer  
Dim strDatei As String  
If Right(strPath, 1) <> "\" Then strPath = strPath & "\"  
strDatei = Dir(strPath & strPattern)  
Do While strDatei <> ""  
    intCounter = intCounter + 1  
ReDim Preserve arrDateien(1 To intCounter)  
    arrDateien(intCounter) = strDatei  
    strDatei = Dir()  
Loop  
If intCounter = 0 Then  
    ReDim arrDateien(1)  
    arrDateien(1) = False
```

```
End If  
FileArray = arrDateien  
End Function
```

21.4. Sortieren der Dateien eines Verzeichnisses nach Dateiname

```
Sub CallQuickSortFilesA()  
    Dim arr As Variant  
    Dim intCounter As Integer  
    Dim strPath As String  
    strPath = GetDirectory("Bitte Verzeichnis auswählen:")  
    If strPath = " " Then Exit Sub  
    arr = FileArray(strPath, "*.*")  
    If arr(1) = False Then  
        Beep  
        MsgBox "Keine Dateien gefunden!"  
        Exit Sub  
    End If  
    QuickSort arr
```

```

Columns("A:B").ClearContents
For intCounter = 1 To UBound(arr)
    Cells(intCounter, 1) = arr(intCounter)
Next intCounter
Columns(1).AutoFit
End Sub

```

21.5. Sortieren der Dateien eines Verzeichnisses nach Dateidatum

```

Sub CallQuickSortFilesB()
    Dim arrDate() As Variant
    Dim arr As Variant
    Dim intCounter As Integer
    Dim strPath As String
    strPath = GetDirectory("Bitte Verzeichnis auswählen:")
    If strPath = "" Then Exit Sub
    arr = FileArray(strPath, "**.*")
    If arr(1) = False Then
        Beep

```

```

MsgBox "Keine Dateien gefunden!"
Exit Sub
End If
Columns("A:B").ClearContents
ReDim arrDate(1 To 2, 1 To UBound(arr))
For intCounter = 1 To UBound(arr)
    arrDate(1, intCounter) = arr(intCounter)
    arrDate(2, intCounter) = FileDateTime(strPath &
arr(intCounter))
Next intCounter
Columns(1).ClearContents
For intCounter = 1 To UBound(arr)
    Cells(intCounter, 1) = arrDate(1, intCounter)
    Cells(intCounter, 2) = arrDate(2, intCounter)
Next intCounter
Range("A1").CurrentRegion.Sort key1:=Range("B1"), -
    order1:=xlAscending, header:=xlNo
Columns("A:B").AutoFit
End Sub

```

21.6. Sortieren der Arbeitsblätter der aktiven Arbeitsmappe

```
Sub CallQuickSortWks ()
    Dim arr() As String
    Dim intCounter As Integer
    ReDim arr(1 To Worksheets.Count)
    For intCounter = 1 To Worksheets.Count
        arr(intCounter) = Worksheets(intCounter).Name
    Next intCounter
    QuickSort arr
    For intCounter = UBound(arr) To 1 Step -1
        Worksheets(arr(intCounter)).Move before:=Worksheets(1)
    Next intCounter
End Sub
```

21.7. Sortieren einer Tabelle nach einer benutzerdefinierten Sortierfolge

```
Sub SortBasedOnCustomList ()  
    Application.AddCustomList ListArray:=Range ("B2:B14")  
    Range ("A16:B36").Sort _  
        key1:=Range ("B17"), _  
        order1:=xlAscending, _  
        header:=xlYes, _  
        OrderCustom:=Application.CustomListCount + 1  
    Application.DeleteCustomList Application.CustomListCount  
End Sub
```

21.8. Sortieren einer Datums-Tabelle ohne Einsatz der Excel-Sortierung

```
Sub CallQuickSortDate ()
```



```

Dim arr(1 To 31) As Date
Dim intRow As Integer
For intRow = 2 To 32
    arr(intRow - 1) = Cells(intRow, 1)
Next intRow
Call QuickSort(arr)
For intRow = 2 To 32
    Cells(intRow, 1).Value = arr(intRow - 1)
Next intRow
End Sub

```

21.9. Sortieren einer Tabelle nach sechs Sortierkriterien

```

Sub SortSixColumns()
    Dim intCounter As Integer
    For intCounter = 2 To 1 Step -1
        Range("A1").CurrentRegion.Sort _
            key1:=Cells(1, intCounter * 3 - 2), _
            order1:=xlAscending, _

```

```

key2:=Cells(1, intCounter * 3 - 1), _
order2:=xlAscending, _
key3:=Cells(1, intCounter * 3), _
order3:=xlAscending, _
header:=xlNo
Next intCounter
End Sub

```

21.10. Sortieren mit Ae vor Ä und Sch vor S

```

Sub SpecialSort ()
    With Columns("A")
        .Replace What:="Ä", Replacement:"Ae", LookAt:=xlPart,
SearchOrder _
            :=xlByRows, MatchCase:=True
    Sch", Replacement:="Rzz", LookAt:=xlPart, _
        SearchOrder:=xlByRows, MatchCase:=True
        .Sort key1:=Range("A2"), order1:=xlAscending, header:=xlGuess,
_
        OrderCustom:=1, MatchCase:=False,
Orientation:=xlTopToBottom

```

```

.Replace What:="Rzz", Replacement:="Sch", LookAt:=xlPart, _
SearchOrder:=xlByRows, MatchCase:=True
.Replace What:="Ae", Replacement:="Ä", LookAt:=xlPart,
SearchOrder _
:=xlByRows, MatchCase:=True
End With
End Sub

```

21.11. Sortieren nach der Häufigkeit des Vorkommens

21.12. Sortieren einschließlich der ausgeblendeten Zeilen

```

Sub SortAll()
Dim rngHidden As Range
Dim lngLastRow As Long, lngRow As Long
Application.ScreenUpdating = False
Set rngHidden = Rows(1)

```

```

lngLastRow = Cells(Rows.Count, 1).End(xlUp).Row
For lngRow = 1 To lngLastRow
    If Rows(lngRow).Hidden = True Then
        Set rngHidden = Union(rngHidden, Rows(lngRow))
    End If
Next lngRow
rngHidden.EntireRow.Hidden = False
Range("A1").CurrentRegion.Sort key1:=Range("A2"), _
    order1:=xlAscending, header:=xlYes
rngHidden.EntireRow.Hidden = True
Rows(1).Hidden = False
Application.ScreenUpdating = True
End Sub

```

21.13. Sortieren mehrerer Tabellenblattbereiche

```

Sub MultiSort()
    Dim intRow As Integer
    For intRow = 1 To 19 Step 6
        Range(Cells(intRow, 1), Cells(intRow + 4, 8)).Sort _
            key1:=Cells(intRow + 1, 7), _

```

```
order1:=xlAscending, header:=xlYes  
Next intRow  
End Sub
```

21.14. Direkter Aufruf des Sortierdialogs

```
Sub CallSortDialogA()  
Application.Dialogs(xlDialogSort).Show  
End Sub
```

21.15. Aufruf des Sortierdialogs unter Einsatz der Sortier-Schaltfläche

```
Sub CallSortDialogB()  
Range("A1").Select
```

```
CommandBars.FindControl(ID:=928).Execute
End Sub
```

21.16. Sortieren per Matrixfunktion

Author: *Stefan Karrmann*

```
Function MatrixSort(ByRef arr As Variant, ByVal column As Long) As
Variant ()
    MatrixSort = arr.Value2
    Call QuickSortCol(MatrixSort, column)
End Function

Sub QuickSortCol(ByRef VA_array, Optional ByVal column As Long, _
    ' On Error Resume Next
    Dim V_Low2, V_high2, V_loop As Integer
    Dim V_vall As Variant
    Dim tmp As Variant
    Dim ColLow As Long, colHigh As Long, col As Long
```

```

If IsMissing(column) Then
    column = 1
End If

ColLow = LBound(VA_array, 2)
colHigh = UBound(VA_array, 2)
If IsMissing(V_Low1) Then
    V_Low1 = LBound(VA_array, 1)
End If
If IsMissing(V_high1) Then
    V_high1 = UBound(VA_array, 1)
End If
V_Low2 = V_Low1
V_high2 = V_high1
V_val1 = VA_array((V_Low1 + V_high1) / 2, column)
While (V_Low2 <= V_high2)
    While (V_Low2 < V_high1 -
        And VA_array(V_Low2, column) < V_val1)
        V_Low2 = V_Low2 + 1
    Wend
    While (V_high2 > V_Low1 -
        And VA_array(V_high2, column) > V_val1)
        V_high2 = V_high2 - 1
    Wend

```

```

If (V_Low2 <= V_high2) Then
  For col = ColLow To colHigh
    tmp = VA_array(V_Low2, col)
    VA_array(V_Low2, col) = VA_array(V_high2, col)
    VA_array(V_high2, col) = tmp
  Next col
  V_Low2 = V_Low2 + 1
  V_high2 = V_high2 - 1
End If
Wend
If (V_high2 > V_Low1) Then Call _
  QuickSortCol(VA_array, column, V_Low1, V_high2)
If (V_Low2 < V_high1) Then Call _
  QuickSortCol(VA_array, column, V_Low2, V_high1)
End Sub

```

21.17. Stringfolge sortieren

Author: *Markus Wilmes*


```

Sub DemoSort ()
    Dim strSort As String
    strSort = "ak dv ad sf ad fa af dd da fa dl 25 24 ad fx "
    Call QuickSortStr(strSort, 3)
    MsgBox strSort
End Sub

Sub QuickSortStr(ByRef strToSort As String, Optional ByVal lngLen,
Optional ByVal lngLow, Optional ByVal lngHigh)
    Dim lngCLow As Long
    Dim lngCHigh As Long
    Dim lngPos As Long
    Dim varA As Variant
    Dim varB As Variant
    If IsMissing(lngLen) Then
        lngLen = 1
    End If
    If IsMissing(lngLow) Then
        lngLow = 0
    End If
    If IsMissing(lngHigh) Then
        lngHigh = (Len(strToSort) / lngLen) - 1
    End If
    lngCLow = lngLow

```

```

lngCHigh = lngHigh
lngPos = Int((lngLow + lngHigh) / 2)
varA = Mid(strToSort, (lngPos * lngLen) + 1, lngLen)
While (lngCLOW <= lngCHigh)
While (Mid(strToSort, (lngCLOW * lngLen) + 1, lngLen) < varA
And lngCLOW < lngHigh)
    lngCLOW = lngCLOW + 1
Wend
While (Mid(strToSort, (lngCHigh * lngLen) + 1, lngLen) > varA
And lngCHigh > lngLow)
    lngCHigh = lngCHigh - 1
Wend
If (lngCLOW <= lngCHigh) Then
    varB = Mid(strToSort, (lngCLOW * lngLen) + 1, lngLen)
    Mid(strToSort, (lngCLOW * lngLen) + 1, lngLen) =
    Mid(strToSort, (lngCHigh * lngLen) + 1, lngLen)
    Mid(strToSort, (lngCHigh * lngLen) + 1, lngLen) = varB
    lngCLOW = lngCLOW + 1
    lngCHigh = lngCHigh - 1
End If
Wend
If (lngCHigh > lngLow) Then
    Call QuickSortStr(strToSort, lngLen, lngLow, lngCHigh)
End If
If (lngCLOW < lngHigh) Then

```

```
    Call QuickSortStr(strToSort, lngLen, lngCLow, lngHigh)
End If
End Sub
```


22. Beispiele für Schleifen

Siehe auch: ../_Schleifen¹

22.1. Allgemeines / Einleitung

Schleifen sind zentraler Bestandteil jeder Programmiersprache. Anhand von Schleifen ist es möglich, Programmanweisungen mehrmals hintereinander zu

wiederholen.

Beispiel einer Programmierung ohne Schleifeneinsatz:

```
Cells(1, 1).Value = "ZEILE 1"  
Cells(2, 1).Value = "ZEILE 2"  
Cells(3, 1).Value = "ZEILE 3"  
Cells(4, 1).Value = "ZEILE 4"  
Cells(5, 1).Value = "ZEILE 5"  
Cells(6, 1).Value = "ZEILE 6"
```

Beispiel der gleichen Programmierung mit Schleifeneinsatz:

```
For iCounter = 1 To 6  
    Cells(iCounter, 1).Value = "Zeile " & iCounter  
Next iCounter
```

Unter anderem kann der Codeumfang somit erheblich reduziert werden, wie im vorhergehenden Beispiel zu sehen ist. Weitere Vorteile werden anhand der

unterschiedlichen Schleifenarten ersichtlich (z.B. variable Anzahl an Durchläufen). Grundsätzlich gibt es zwei Arten von Schleifen, die Zählschleifen (die Anzahl der Schleifendurchläufe wird durch eine Variable oder konstante Zahl bestimmt) und Prüfschleifen (die Schleife wird durchlaufen solange ein Bedingung wahr bzw. falsch ist).

Grundlagenwissen zu Schleifen lässt sich hier nachlesen: Wikipedia: Schleifen (Programmierung)²

22.2. Schleifentypen-Beispiele

Jeder Schleifentyp kann weitere Bedingungsprüfungen enthalten. Bei Zählschleifen kann die Schrittgröße festgelegt werden; der Default-Wert ist 1.

2 http://de.wikipedia.org/wiki/Schleife_%28Programmierung%29

22.2.1. Zählschleifen

For-To-Next-Schleife

- Prozedur: ForNextCounter
- Art: Sub
- Modul: Standardmodul
- Zweck: Zähler hochzählen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleifenbeginn
 - Wert berechnen und addieren
 - Schleifenende
 - Ergebnisausgabe
- Code:

```
Sub ForNextCounter()  
    Dim dValue As Double
```



```
Dim iCounter As Integer  
For iCounter = 1 To 100  
    dValue = dValue + iCounter * 1.2  
Next iCounter  
MsgBox "Ergebnis: " & dValue  
End Sub
```

For...To...Next-Schleife mit Schrittgrößenangabe nach vorn

- Prozedur: ForNextStepForward
- Art: Sub
- Modul: Standardmodul
- Zweck: Zähler schrittweise hochzählen
- Ablaufbeschreibung:
- Variablendeklaration
 - Schleifenbeginn
 - Wert ausgeben
 - Schleifenende
- Code:

```
Sub ForNextStepForward()  
    Dim iCounter As Integer  
    For iCounter = 1 To 10 Step 2  
        MsgBox iCounter  
    Next iCounter  
End Sub
```

For...To...Next-Schleife mit Schrittgrößenangabe zurück

- Prozedur: ForNextStepBack
- Art: Sub
- Modul: Standardmodul
- Zweck: Zähler schrittweise hochzählen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleifenbeginn
 - Wert ausgeben
 - Schleifenende

- Code:

```
Sub ForNextStepBack ()  
    Dim iCounter As Integer  
    For iCounter = 10 To 1 Step -2  
        MsgBox iCounter  
    Next iCounter  
End Sub
```

22.2.2. Schleifen mit vorangestellte Bedingungsprüfung

While ... Wend-Schleife

- Prozedur: WhileWend
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:

- Variablendeklaration
- Startwert setzen
- Schleifenbeginn
- Wert berechnen und addieren
- Zeilenzähler hochzählen
- Schleifenende
- Wert ausgeben
- Code:

```
Sub WhileWend()  
  Dim iRow As Integer  
  Dim dValue As Double  
  iRow = 1  
  While Not IsEmpty(Cells(iRow, 1))  
    dValue = dValue + Cells(iRow, 1).Value * 1.2  
    iRow = iRow + 1  
  Wend  
  MsgBox "Ergebnis: " & dValue  
End Sub
```

Do ... Loop-Schleife

- Prozedur: DoLoop
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Startwert setzen
 - Schleifenbeginn
 - Wert berechnen und addieren
 - Bedingung prüfen
 - Zeilenzähler hochzählen
 - Schleifenende
 - Wert ausgeben
- Code:

Sub DoLoop()

```

Dim iRow As Integer
Dim dValue As Double
iRow = 1
Do
    dValue = dValue + Cells(iRow, 1).Value * 1.2
If IsEmpty(Cells(iRow + 1, 1)) Then Exit Do
    iRow = iRow + 1
Loop
MsgBox "Ergebnis: " & dValue
End Sub

```

Do ... While-Schleife

- Prozedur: Do While
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Startwert setzen

- Schleifenbeginn mit Bedingung
- Wert berechnen und addieren
- Zeilenzähler hochzählen
- Schleifenende
- Wert ausgeben
- Code:

```
Sub DoWhile()  
    Dim iRow As Integer  
    Dim dValue As Double  
    iRow = 1  
    Do While Not IsEmpty(Cells(iRow, 1))  
        dValue = dValue + Cells(iRow, 1).Value * 1.2  
        iRow = iRow + 1  
    Loop  
    MsgBox "Ergebnis: " & dValue  
End Sub
```

Do-Until-Schleife

- Prozedur: DoUntil
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Startwert setzen
 - Schleifenbeginn mit Bedingung
 - Wert berechnen und addieren
 - Zeilenzähler hochzählen
 - Schleifenende
 - Wert ausgeben
- Code:

```
Sub DoUntil ()
```

```
    Dim iRow As Integer
```



```

Dim dValue As Double
iRow = 1
Do Until IsEmpty(Cells(iRow, 1))
    dValue = dValue + Cells(iRow, 1).Value * 1.2
    iRow = iRow + 1
Loop
MsgBox "Ergebnis: " & dValue
End Sub

```

22.2.3. Schleifen mit nachgestellter Bedingungsprüfung

Do-Until-Schleife

- Prozedur: DoUntil
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration

- Startwert setzen
- Schleifenbeginn
- Wert berechnen und addieren
- Zeilenzähler hochzählen
- Schleifenende mit Bedingung
- Wert ausgeben
- Code:

```

Sub DoLoopWhile ()
    Dim iRow As Integer
    Dim dValue As Double
    iRow = 1
    Do
        dValue = dValue + Cells(iRow, 1).Value * 1.2
        iRow = iRow + 1
    Loop While Not IsEmpty(Cells(iRow - 1, 1))
    MsgBox "Ergebnis: " & dValue
End Sub

```

22.2.4. Weitere Schleifen mit nachgestellter Bedingungsprüfung

Do-Loop-Until-Schleife

- Prozedur: DoLoopUntil
- Art: Sub
- Modul: Standardmodul
- Zweck: Zellen durchlaufen und Einzelwerte berechnen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Startwert setzen
 - Schleifenbeginn
 - Wert berechnen und addieren
 - Zeilenzähler hochzählen
 - Schleifenende mit Bedingung
 - Wert ausgeben
- Code:

```

Sub DoLoopUntil()
    Dim iRow As Integer
    Dim dValue As Double
    iRow = 1
    Do
        dValue = dValue + Cells(iRow, 1).Value * 1.2
        iRow = iRow + 1
    Loop Until IsEmpty(Cells(iRow, 1))
    MsgBox "Ergebnis: " & dValue
End Sub

```

22.3. Objektbezogene Beispiele

22.3.1. Einsatz bei Arbeitsmappen- und Tabellenobjekte

Ausgabe der Arbeitsblattnamen der aktiven Arbeitsmappe

- Prozedur: EachWks

- **Art:** Sub
- **Modul:** Standardmodul
- **Zweck:** Arbeitsblattnamen der aktiven Arbeitsmappe ausgeben
- **Ablaufbeschreibung:**
 - Variablendeklaration
 - Schleifenbeginn
 - Ausgabe der Namen
 - Schleifenende
- **Code:**

```
Sub EachWks()  
    Dim wks As Worksheet  
    For Each wks In Worksheets  
        MsgBox wks.Name  
    Next wks  
End Sub
```

Ausgabe der Arbeitsblattnamen alle geöffneten Arbeitsmappen

- Prozedur: EachWkbWks
- Art: Sub
- Modul: Standardmodul
- Zweck: Arbeitsblattnamen aller geöffneten Arbeitsmappe ausgeben
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleifenbeginn Arbeitsmappen
 - Schleifenbeginn Arbeitsblätter
 - Ausgabe der Namen
 - Schleifenende Arbeitblätter
 - Schleifenende Arbeitsmappen
- Code:

Sub EachWkbWks ()

Dim wkb **As** Workbook

Dim wks **As** Worksheet

```
For Each wkb In Workbooks
For Each wks In wkb.Worksheets
    MsgBox wkb.Name & vbCrLf & " -" & wks.Name
Next wks
Next wkb
End Sub
```

Ausgabe der integrierten Dokumenteneigenschaften der aktiven Arbeitsmappe

- Prozedur: EachWkbWks
- Art: Sub
- Modul: Standardmodul
- Zweck: Integrierte Dokumenteneigenschaften der aktiven Arbeitsmappe ausgeben
- Ablaufbeschreibung:
 - Variablendeklaration
 - Fehlerroutine
 - Schleifenbeginn

- Ausgabe der Namen
- Schleifenende
- Ende der Fehlerroutine
- Code:

```

Sub EachDPWkb ()
    Dim oDP As DocumentProperty
    On Error Resume Next
    For Each oDP In ThisWorkbook.BuiltinDocumentProperties
        MsgBox oDP.Name & " : " & oDP.Value
    Next oDP
    On Error GoTo 0
End Sub

```

Ausgabe der Formatvorlagen der aktiven Arbeitsmappe

- Prozedur: EachWkbWks
- Art: Sub
- Modul: Standardmodul

- Zweck: Formatvorlagen der aktiven Arbeitsmappe ausgeben
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleifenbeginn
 - Wert ausgeben
 - Schleifenende
- Code:

```
Sub EachStylesWkb()  
  Dim oStyle As Style  
  For Each oStyle In wkb.Styles  
    MsgBox oStyle.Name  
  Next oStyle  
End Sub
```

Ausgabe der einzelnen Zelladressen eines vorgegebenen Bereiches

- Prozedur: EachWkbWks

- Art: Sub
- Modul: Standardmodul
- Zweck: Zelladressen eines vorgegebenen Bereiches ausgeben
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleifenbeginn
 - Wert ausgeben
 - Schleifenende
- Code:

```

Sub EachCellWks ()
  Dim rng As Range
  For Each rng In Range ("A1:B2")
    MsgBox rng.Address (rowabsolute:=False, columnabsolute:=False)
  Next rng
End Sub

```

22.3.2. Einsatz bei tabellenintegrierten Steuerelement-Objekten

Prüfung, welches Optionsfeld in einer vorgegebenen Gruppe von Optionsfeldgruppen aktiviert ist

- Prozedur: EachWks
- Art: Sub
- Modul: Klassenmodul der Tabelle
- Zweck: Ausgabe des Namens des aktivierten Optionsfelds einer vorgegebenen Optionsfeldgruppe
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleife über alle Steuerelemente der Tabelle
 - Prüfung des Typnamens des Steuerelements
 - Wenn es sich um ein Optionsfeld handelt...
 - Übergabe an eine Objektvariable
 - Wenn das Optionsfeld aktiviert ist und es sich um ein Steuerelement von der Gruppe GroupB handelt...

- Ausgabe des Namens des Steuerelements
- Schleifenende
- Code:

```

Sub IfSelected()
    Dim oOle As OLEObject
    Dim oOpt As msforms.OptionButton
    For Each oOle In OLEObjects
        If TypeName(oOle.Object) = "OptionButton" Then
            Set oOpt = oOle.Object
            If oOpt And oOpt.GroupName = "GroupB" Then
                MsgBox "In GroupB ist " & oOpt.Caption & " aktiviert"
            End If
        End If
    Next oOle
End Sub

```

22.3.3. Einsatz bei Userform-Steuer-element-Objekten

Prüfung, welche CheckBox-Elemente einer UserForm aktiviert sind

- Prozedur: cmdRead_Click
- Art: Sub
- Modul: Klassenmodul der UserForm
- Zweck: Ausgabe des Namens aktivierter CheckBox-Elemente einer UserForm
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleife über alle Steuer-elemente der UserForm
 - Wenn es sich um eine CheckBox handelt...
 - Wenn die CheckBox aktiviert ist...
 - Einlesen des CheckBox-Namens in eine String-Variable
 - Schleifenende
 - Wenn keine aktivierte CheckBoxes gefunden wurden...
 - Negativmeldung

- Sonst...
- Ausgabe des oder der Namen der aktivierten CheckBoxes
- Code:

```

Private Sub cmdRead_Click()
    Dim oCntr As msforms.Control
    Dim sMsg As String
    For Each oCntr In Controls
        If TypeName(oCntr) = "CheckBox" Then
            If oCntr Then
                sMsg = sMsg & " " & oCntr.Name & vbCrLf
            End If
        End If
    Next oCntr
    If sMsg = "" Then
        MsgBox "Es wurde keine CheckBox aktiviert!"
    Else
        MsgBox "Aktivierte CheckBoxes:" & vbCrLf & sMsg
    End If
End Sub

```

Bedingtes Einlesen von ListBox-Elementen in eine zweite ListBox

- Prozedur: cmdAction_Click
- Art: Sub
- Modul: Klassenmodul der UserForm
- Zweck: Ausgabe des Namens aktivierter CheckBox-Elemente einer UserForm
- Ablaufbeschreibung:
 - Variablendeklaration
 - Schleife über alle Listelemente des ersten Listenfelds
 - Wenn das Listenelement den Bedingungen entspricht...
 - Übergabe an das zweite Listenfeld
 - Schleifenende
- Code:

```
Private Sub cmdAction_Click()  
Dim iCounter As Integer  
For iCounter = 0 To lstAll.ListCount - 1
```

```
If CDate(lstAll.List(iCounter)) &gt;= CDate(txtStart) And _  
    CDate(lstAll.List(iCounter)) &lt;= CDate(txtEnd) Then  
    lstFilter.AddItem lstAll.List(iCounter)  
End If  
Next iCounter  
End Sub
```


23. Rechtschreibprüfung

23.1. Die CheckSpelling-Methode

Die **CheckSpelling**-Methode kann aufgerufen werden mit:

- **Syntax1:** *Ausdruck*.CheckSpelling([CustomDictionary], [IgnoreUppercase], [AlwaysSuggest], [SpellLanguage])

- **CustomDictionary:** Das Benutzer-Wörterbuch (optional) Eingerichtet sind zwei (am Anfang leere) Wörterbücher:

- **BENUTZER.DIC** für die deutsche Sprachversion
 - **custom.dic** für die englische Sprachversion
- Neue Wörterbücher können hinzugefügt werden.

- **IgnoreUppercase:** Groß/Kleinschreibung ignorieren (optional)

- **AllwaysSuggest:** Schreibweise vorschlagen (optional)

- **Sprache:** Die zugrundeliegende Sprache

Die möglichen Sprachversionen ergeben sich aus dem Rechtschreibungs-Dialog und sind in der Regel:

- Deutsch (Deutschland)
- Deutsch (Österreich)
- Deutsch (Schweiz)

- Englisch (Australien)
- Englisch (Großbritannien)
- Englisch (Kanada)
- Englisch (USA)
- Französisch (Frankreich)
- Französisch (Kanada)
- Italienisch (Italien)
- **Syntax2:**

Ausdruck.CheckSpelling(Word, [CustomDictionary], [IgnoreUppercase])

Word: Der zu prüfende Begriff

Wird als *Ausdruck Application* vorgegeben, kommt Syntax 2 zur Anwendung.

23.2. Wort prüfen

- Prozedur: CheckWord

- Art: Sub
- Modul: Standardmodul
- Zweck: Einzelwort prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Fehlerroutine initialisieren
 - Prüfbegriff festlegen
 - Wenn der Prüfbegriff nicht gefunden wurde...
 - Negativmeldung
 - Sonst...
 - Positivmeldung
 - Prozedur beenden
 - Start Fehleroutine
 - Fehlermeldung
- Code:

Sub CheckWord()

Dim sWorth **As** String

```

On Error GoTo ERRORHANDLER
sWorth = Range("A1").Value
If Not Application.CheckSpelling( _
    word:=sWorth, _
    customdictionary:="BENUTZER.DIC", _
    ignoreuppercase:=False) Then
    MsgBox "Keine Entsprechung für das Wort " & sWorth & "
    gefunden!"
Else
    MsgBox "Das Wort " & sWorth & " ist vorhanden!"
End If
Exit Sub
ERRORHANDLER:
    Beep
    MsgBox _
        prompt:="Die Rechtschreibprüfung ist nicht installiert!"
End Sub

```

23.3. Wort auf englisch prüfen

- Prozedur: SpellLanguage

- Art: Sub
- Modul: Standardmodul
- Zweck: Englisches Einzelwort prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Aktuelle Spracheinstellung einlesen
 - Wenn es sich um die Excel-Version 7.0 handelt zum 1. Errorhandler springen
 - Initialisierung des 2. Errorhandlers
 - Prüfbegriff einlesen
 - Wenn der Prüfbegriff nicht im kanadisch-englischen Wörterbuch gefunden wurde...
 - Negativmeldung
 - Sonst...
 - Positivmeldung
 - Prüfsprache auf aktuelle Office-Spracheinstellung setzen
 - Prozedur beenden
 - Erster Errorhandler

- Zweiter Errorhandler
- Code:

```

Sub SpellLanguage ()
    Dim lLang As Long
    Dim sWorth As String
    Dim bln As Boolean
    lLang = Application.LanguageSettings.LanguageID (msoLanguageIDUI)
    If Left (Application.Version, 1) = "7" Then GoTo ERRORHANDLER1
    On Error GoTo ERRORHANDLER2
    sWorth = Range ("A2").Value
    If Not Range ("A2").CheckSpelling ( _
        customdictionary:="BENUTZER.DIC", _
        ignoreuppercase:=False, _
        spelllang:=3081) Then
        MsgBox "Keine Entsprechung für das Wort " & sWorth & "
        gefunden!"
    Else
        MsgBox "Das Wort " & sWorth & " ist entweder vorhanden" & vbCrLf
        & _
            "oder es wurde keine Korrektur gewünscht!"
    End If
    bln = Range ("A2").CheckSpelling ("Test", spelllang:=lLang)
    Exit Sub

```

```
ERRORHANDLER1:
  MsgBox "Die Sprachfestlegung ist erst ab XL9 möglich!"
Exit Sub
ERRORHANDLER2:
  Beep
  MsgBox _
    prompt:="Die Rechtschreibprüfung ist nicht installiert!"
End Sub
```

23.4. Steuerelement-TextBox prüfen

Bitte beachten: OLEObjekte lassen sich nicht über die **CheckSpelling-** Methode ansprechen, ihre Texte müssen ausgelesen werden.

- Prozedur: CheckTextBoxA
- Art: Sub
- Modul: Standardmodul
- Zweck: Den Inhalt einer TextBox aus der Steuerelement-ToolBox prüfen
- Ablaufbeschreibung:

- Variablendeklaration
- Eine Schleife über alle OLEObjekte des aktiven Blattes bilden
- Wenn es sich um eine TextBox handelt...
- TextBox-Inhalt in eine String-Variable einlesen
- Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
- Eine Schleife über alle Einzelwörter bilden
- Wenn das Wort nicht gefunden wurde...
- Negativmeldung
- Code:

```

Sub CheckTextBoxA()
    Dim oTxt As OLEObject
    Dim arrWrd() As String, sTxt As String
    Dim iCounter As Integer
    For Each oTxt In ActiveSheet.OLEObjects
        If TypeOf oTxt Object Is MSForms.TextBox Then
            sTxt = oTxt.Object.Text
            arrWrd = MySplit(sTxt, " ")
    
```

```

For iCounter = 1 To UBound(arrWrd)
  If Not Application.CheckSpelling( _
    word:=arrWrd(iCounter), _
    customdictionary:="BENUTZER.DIC", _
    ignoreuppercase:=False) Then
    MsgBox arrWrd(iCounter) & " aus der TextBox " _
      & oTxt.Name & " wurde nicht im Wörterbuch
gefunden!"
  End If
Next iCounter
End If
Next oTxt
End Sub

```

23.5. Zeichnen-TextBox global prüfen

- Prozedur: CheckTxtBoxB
- Art: Sub
- Modul: Standardmodul

- Zweck: Den Inhalt einer TextBox aus der Zeichen-Symbolleiste global prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Wenn alle Wörter des TextBox-Inhalts gefunden wurden...
 - Positivmeldung
 - Sonst...
 - Negativmeldung
- Code:

```

Sub CheckTextBoxB ()
  If Application.CheckSpelling( _
    word:=ActiveSheet.TextBoxes("txtSpelling").Text, _
    customdictionary:="BENUTZER.DIC", _
    ignoreuppercase:=False) Then
    MsgBox "Alle Wörter wurden gefunden!"
  Else
    MsgBox "Mindestens ein Wort wurde nicht gefunden!"
  End If
End Sub

```

23.6. Zeichnen-TextBox einzeln prüfen

Bitte beachten: OLEObjekte lassen sich nicht über die **CheckSpelling**-Methode ansprechen, ihre Texte müssen ausgelesen werden.

- Prozedur: CheckTextBoxC
- Art: Sub
- Modul: Standardmodul
- Zweck: Alle Wörter aus einer TextBox aus der Zeichnen-Symbolleiste einzeln prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - TextBox-Inhalt in eine String-Variable einlesen
 - Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
- Eine Schleife über alle Einzelwörter bilden
- Wenn das Wort nicht gefunden wurde...

- Negativmeldung
- Code:

```

Sub CheckTextBoxC ()
    Dim arrWrd() As String, sTxt As String
    Dim iCounter As Integer
    sTxt = ActiveSheet.TextBoxes("txtSpelling").Text
    arrWrd = MySplit(sTxt, " ")
    For iCounter = 1 To UBound(arrWrd)
        If Not Application.CheckSpelling( _
            word:=arrWrd(iCounter), _
            customdictionary:="BENUTZER.DIC", _
            ignoreuppercase:=False) Then
            MsgBox arrWrd(iCounter) & " aus der TextBox " & _
                "txtSpelling wurde nicht im Wörterbuch gefunden!"
        End If
    Next iCounter
End Sub

```

23.7. Zellbereich prüfen

- Prozedur: CheckRange
- Art: Sub
- Modul: Standardmodul
- Zweck: Einen Zellbereich global prüfen
- Ablaufbeschreibung:
 - Wenn alle Wörter eines Bereiches gefunden wurden...
 - Positivmeldung
 - Sonst...
 - Negativmeldung
- Code:

```
Sub CheckRange ()  
    If Range ("A4:A8").CheckSpelling Then  
        MsgBox "Entweder alle Wörter wurden gefunden" & vbCrLf & _  
            "oder es wurde keine Korrektur gewünscht!"  
    Else  
        MsgBox "Es wurden nicht alle Wörter aus dem Bereich A4:A8
```

gefunden!"

End If

End Sub

23.8. Gültigkeitsfestlegungen prüfen

- Prozedur: CheckValidation
- Art: Sub
- Modul: Standardmodul
- Zweck: Eingabe- und Fehlermeldungstexte einer Gültigkeitsfestlegung prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - Zelle mit Gültigkeitsprüfung an eine Objektvariable übergeben
 - Wenn die Zelle eine Gültigkeitsprüfung enthält...
 - Fehlermeldungs-Text in Stringvariable einlesen
 - Wenn eine Fehlermeldung festgelegt wurde...

- Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
- Eine Schleife über alle Wörter bilden
- Wenn das jeweilige Wort nicht gefunden wurde...
- Negativmeldung
- Eingabe-Text in Stringvariable einlesen
- Wenn ein Eingabetext festgelegt wurde...
- Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
- Eine Schleife über alle Wörter bilden
- Wenn das jeweilige Wort nicht gefunden wurde...
- Negativmeldung
- Code:

Sub CheckValidation ()

Dim rng **As** Range


```

Dim arrWrd() As String, sTxt As String
Dim iCounter As Integer
Set rng = Range("A10")
If Abs(rng.Validation.Type) >= 0 Then
    sTxt = rng.Validation.ErrorMessage
    If sTxt &lt;&gt; vbNullString Then
        arrWrd = MySplit(sTxt, " ")
        For iCounter = 1 To UBound(arrWrd)
            If Not Application.CheckSpelling( _
                word:=arrWrd(iCounter), _
                customdictionary:="BENUTZER.DIC", _
                ignoreuppercase:=False) Then
                MsgBox arrWrd(iCounter) & " aus der Fehlermeldung " &
                    "wurde nicht im Wörterbuch gefunden!"
            End If
        Next iCounter
    End If
    sTxt = rng.Validation.InputMessage
    Erase arrWrd
    If sTxt &lt;&gt; vbNullString Then
        arrWrd = MySplit(sTxt, " ")
        For iCounter = 1 To UBound(arrWrd)
            If Not Application.CheckSpelling( _
                word:=arrWrd(iCounter), _

```

```

customdictionary:="BENUTZER.DIC", _
ignoreuppercase:=False) Then
MsgBox arrWrd(iCounter) & " aus der Eingabemeldung " &
_
    "wurde nicht im Wörterbuch gefunden!"
End If
Next iCounter
End If
End If
End Sub

```

23.9. UserForm-TextBox prüfen

- Prozedur: cmdSpelling_Click
- Art: Sub
- Modul: Klassenmodul der UserForm
- Zweck: Inhalt einer UserForm-TextBox prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - TextBox-Text in eine String-Variablen einlesen

- Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
- Schleife über alle Wörter bilden
- Wenn das jeweilige Wort nicht gefunden wurde...
- Negativmeldung
- Schleife verlassen
- Wenn ein Wort nicht gefunden wurde...
- Rahmen mit der TextBox bilden
- Den Focus der TextBox zuordnen
- Erstes Zeichen für die Textmarkierung festlegen
- Länge der Textmarkierung festlegen
- Code:

```

Private Sub cmdSpelling_Click()
Dim arrWrd() As String, sTxt As String, sWhole As String
Dim lChar As Long
Dim iCounter As Integer
sTxt = txtSpelling.Text
sWhole = sTxt

```

```

arrWrd = MySplit(sTxt, " ")
For iCounter = 1 To UBound(arrWrd)
    If Not Application.CheckSpelling( _
        word:=arrWrd(iCounter), _
        customdictionary:="BENUTZER.DIC", _
        ignoreuppercase:=False) Then
        MsgBox arrWrd(iCounter) & " aus der TextBox " & _
            "txtSpelling wurde nicht im Wörterbuch gefunden!"
        lChar = InStr(sWhole, arrWrd(iCounter))
        Exit For
    End If
Next iCounter
If lChar > 0 Then
    With txtSpelling
        .SetFocus
        .SelStart = lChar - 1
        .SelLength = Len(arrWrd(iCounter))
    End With
End If
End Sub

```

23.10. UserForm-TextBox prüfen

- Prozedur: cmdSpelling_Click
- Art: Sub
- Modul: Klassenmodul der UserForm
- Zweck: Inhalt einer UserForm-TextBox prüfen
- Ablaufbeschreibung:
 - Variablendeklaration
 - TextBox-Text in eine String-Variablen einlesen
 - Funktion zum Aufsplitten des Textes in Einzelwörter aufrufen (bei Excel-Versionen ab XL2000 kann hier die VBA-Split-Funktion eingesetzt werden)
 - Schleife über alle Wörter bilden
 - Wenn das jeweilige Wort nicht gefunden wurde...
 - Negativmeldung
 - Schleife verlassen
 - Wenn ein Wort nicht gefunden wurde...
 - Rahmen mit der TextBox bilden

- Den Focus der TextBox zuordnen
- Erstes Zeichen für die Textmarkierung festlegen
- Länge der Textmarkierung festlegen
- Code:

```

Private Sub cmdSpelling_Click()
    Dim arrWrd() As String, sTxt As String, sWhole As String
    Dim lChar As Long
    Dim iCounter As Integer
    sTxt = txtSpelling.Text
    sWhole = sTxt
    arrWrd = MySplit(sTxt, " ")
    For iCounter = 1 To UBound(arrWrd)
        If Not Application.CheckSpelling( _
            word:=arrWrd(iCounter), _
            customdictionary:="BENUTZER.DIC", _
            ignoreuppercase:=False) Then
            MsgBox arrWrd(iCounter) & " aus der TextBox " & _
                "txtSpelling wurde nicht im Wörterbuch gefunden!"
            lChar = InStr(sWhole, arrWrd(iCounter))
            Exit For
        End If
    Next iCounter

```

```
If lChar > 0 Then  
  With txtSpelling  
    .SetFocus  
    .SelStart = lChar - 1  
    .SelLength = Len(arrWrd(iCounter))  
  End With  
End If  
End Sub
```

23.11. Bei Eingabe Rechtschreibprüfung aufrufen

- Prozedur: Worksheet_Change
- Art: Sub
- Modul: Klassenmodul des Arbeitsblattes
- Zweck: Bei Zelleingabe in Spalte A die Rechtschreibprüfung aufrufen
- Ablaufbeschreibung:
 - Wenn die Eingabezelle in Spalte A liegt, dann...
 - Warnmeldungen ausschalten
 - Rechtschreibprüfung aufrufen

- Warnmeldungen einschalten
- Code:

```
Private Sub Worksheet_Change(ByVal Target As Range)
If Target.Column = 1 Then
    Application.DisplayAlerts = False
    Target.CheckSpelling
    Application.DisplayAlerts = True
End If
End Sub
```

23.12. Bei Doppelklick Rechtschreibprüfung aufrufen

- Prozedur: Worksheet_BeforeDoubleClick
- Art: Sub
- Modul: Klassenmodul des Arbeitsblattes
- Zweck: Bei Doppelklick in Spalte B die Rechtschreibprüfung aufrufen
- Ablaufbeschreibung:

- Wenn die Eingabezelle in Spalte B liegt, dann...
- Doppelklick-Voreinstellung ausschalten
- Warnmeldungen ausschalten
- Rechtschreibprüfung aufrufen
- Warnmeldungen einschalten
- Code:

```
Private Sub Worksheet_BeforeDoubleClick ( _  
    ByVal Target As Range, Cancel As Boolean)  
    If Target.Column = 2 Then  
        Cancel = True  
        Application.DisplayAlerts = False  
        Target.CheckSpelling  
        Application.DisplayAlerts = True  
    End If  
End Sub
```

23.13. Beim Schließen jeder Arbeitsmappe eine Rechtschreibprüfung durchführen

Der nachfolgende Code muß in die **Personl.xls** eingegeben werden, damit er für alle nach Sitzungsstart zu öffnenden und zu schließenden Arbeitsmappen Gültigkeit hat.

23.13.1. Im Klassenmodul der Arbeitsmappe:

```
Dim xlApplication As New clsApp

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Set xlApplication.xlApp = Nothing
End Sub

Private Sub Workbook_Open()
    Set xlApplication.xlApp = Application
    Call CreateCmdBar
End Sub
```

23.13.2. In einem Klassenmodul mit dem Namen clsApp:

```
Public WithEvents xlApp As Excel.Application
```

```
Private Sub xlApp_WorkbookBeforeClose(ByVal Wb As Excel.Workbook, _  
Cancel As Boolean)
```

```
    Dim wks As Worksheet
```

```
    For Each wks In Wb.Worksheets  
        wks.CheckSpelling
```

```
    Next
```

```
End Sub
```


Teil VI.

Anhang

24. Weitere Beispiele

24.1. Belegte Zellen bestimmen

Mit dem nachfolgenden Beispiel können die erste und letzte belegte Zelle in einer Zeile bestimmt werden. Klicken Sie eine beliebige Zeile an und starten das Makro. Ein Meldungsfenster gibt Ihnen Auskunft, welches die erste und letzte belegte Zelle der angeklickten Zeile ist.

```
Sub ErsteUndLetzteBelegteZelleInZeile()
```

```
    Dim lngSpalte1&, lngSpalte2 As Long: Dim strAusgabertext As String
```

```
    lngSpalte1 = Cells(ActiveCell.Row, 1).End(xlToLeft).Column
```

```

lngSpalte2 = Cells(ActiveCell.Row,
Rows(ActiveCell.Row).Cells.Count).End(xlToLeft).Column
If IsEmpty(Cells(ActiveCell.Row, 1)) = False Then lngSpalte1 = 1

strAusgabebetext = Switch(lngSpalte1 =
Rows(ActiveCell.Row).Cells.Count And lngSpalte2 = 1, _
    "Zeile " & ActiveCell.Row & " ist leer.", lngSpalte1 >= 1 And
lngSpalte2 > lngSpalte1, _
    "In der angeklickten Zeile ist die erste belegte Zelle " &
Cells(ActiveCell.Row, _
    lngSpalte1).Address(False, False) & vbCrLf & " mit dem Wert " &
_
    Cells(ActiveCell.Row, lngSpalte1) & " und die letzte Zelle
ist " & _
    Cells(ActiveCell.Row, lngSpalte2).Address(False, False) &
vbCrLf & " mit dem Wert " & _
    Cells(ActiveCell.Row, lngSpalte2) & ". ", lngSpalte1 =
lngSpalte2, _
    "Es ist nur Zelle " & Cells(ActiveCell.Row,
lngSpalte1).Address(False, False) & _
    " mit dem Wert " & Cells(ActiveCell.Row, lngSpalte1) & "
belegt.")
MsgBox strAusgabebetext, vbInformation
End Sub

```


24.2. Add-Ins

Add-In installieren

```
Sub InstallAddIn()  
    Dim AddInNeu As AddIn  
    On Error Resume Next  
    Set AddInNeu = AddIns.Add(Filename:=Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) &  
        "\\Microsoft\\AddIns\\neuesAddIn.xlam")  
    AddInNeu.Installed = True  
    MsgBox AddInNeu.Title & " wurde installiert."  
Exit Sub  
ErrorHandler:  
    MsgBox "An error occurred."  
End Sub
```

Add-In deinstallieren

```
Sub AddInEinbinden()  
    Application.AddIns("neuesAddIn").Installed = False  
End Sub
```

Add-In schließen

```
Sub addInSchließen()  
On Error Resume Next  
Workbooks("neuesAddIn.xlam").Close  
End Sub
```

24.3. Variablentyp bestimmen

Klicken Sie eine belegte Zelle eines Arbeitsblatts an. Mit dem Makro können Sie den Variablentyp einer Zelle bestimmen.

```
Sub ZellenWerttypErmitteln()  
Dim strVariablentyp As String  
Dim byteIndex As Byte  
byteIndex = VarType(ActiveCell)  
strVariablentyp = Choose(byteIndex + 1, "Empty", "Null",  
"Integer", "Long", _  
"Single", "Double", "Currency", "Date", "String", "Object",
```

```
"Error", "Boolean")
MsgBox strVariablentyp
End Sub
```

24.4. Arbeitsblattexistenz bestimmen

Mit diesem Makro können Sie die Existenz eines Tabellenblatts überprüfen. Wenn Sie in die zweite Inputbox keinen Mappennamen eintragen, wird unterstellt, dass die Existenz des eingegebenen Tabellenblatts in der aktivierten Mappe geprüft werden soll. (Beachte: der zu überprüfende BlattCodename ist nicht identisch mit dem Tabellennamen (wie auf dem Tabellenregisterblatt). Sie können den jeweiligen BlattCodennamen im Projektextplorer herausfinden. Der Blattcodename ist Tabelle1, Tabelle2 usw.) Verweis: Microsoft Visual Basic for Applications Extensibility

```
Function BlattDa(strBlattCodename As String, Optional Mappe As
Workbook) As Boolean
```

```

If Mappe Is Nothing Then
    Set Mappe = ActiveWorkbook
Else
    For Each Workbook In Application.Workbooks
        If Mappe.Name = Workbook.Name Then Set Mappe = Workbook
    Next Workbook
End If
For Each Worksheet In Mappe.Worksheets
    If Mappe.VBProject.VBComponents(Worksheet.CodeName).Name =
strBlattCodename Then
        BlattDa = True
    End If
Next Worksheet
End Function

Sub CheckForSheet ()
    Dim boolBlattDa As Boolean
    Dim strMappename$
    Dim strBlattCodename$
    strBlattCodename = InputBox("Gebe den Blattcodenamen ein")
    If strBlattCodename = "" Then Exit Sub
    strMappename = InputBox("Gebe den Namen der geöffneten Mappe
ohne Dateierdung ein!" & _
        "Falls Sie nichts eintragen und ok klicken, wird die aktuelle
Mappe geprüft!")

```

```
If strMappenname <> "" Then
    On Error Resume Next
    If Workbooks(strMappenname) Is Nothing Then
        MsgBox "Die Mappe ist nicht geöffnet oder existiert
        nicht", vbCritical
    Exit Sub
End If
End If
If strMappenname = "" Then
    boolBlattDa = BlattDa(strBlattCodename)
Else
    boolBlattDa = BlattDa(strBlattCodename,
    Workbooks(strMappenname))
End If
If boolBlattDa Then
    MsgBox "Das Blatt existiert!"
Else
    MsgBox "The worksheet does NOT exist!"
End If
End Sub
```

24.5. Tabellenlisten mit Anwenderformular editieren

Erzeugen Sie händisch oder per VBA-Makro eine Tabellenliste und fügen das erste Makro in das Codemodul des verwendeten Arbeitsblatts ein.

Danach erstellen Sie ein Anwenderformular Userform1 und platzieren darauf ein Listenfeld, drei Befehlsschaltflächen Commandbutton1 - 3 und für jede zu editierende Tabellenspalte jeweils ein Texteingabefeld TextBox.

CommandButton1 - Caption: Zeile hinzufügen CommandButton2: Caption: Zeile ändern CommandButton3: Caption: Zeile löschen Um das Makro zu starten, klicken Sie doppelt auf die Tabellenliste.

Codemodul des verwendeten Arbeitsblatts

```
Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As
    Boolean)
    Dim strListobjectname
On Error Resume Next
If Selection.ListObject.Name = "" Then
```

```

MsgBox "Keine Tabellenliste angeklickt"
Exit Sub
Else
    strListobjectname = Selection.ListObject.Name
End If
Load UserForm1
With UserForm1
    .Caption = "Verkaufsliste"
    .Show
End With
End Sub

```

Codemodul des Anwenderformulars, Name: Userform1

```

Private strListobjectname$
Sub ListenfeldFüllen()
    Dim i%, intSpaltenzahl%, sngSpaltenbreite!, varSpaltenbreiten
    intSpaltenzahl =
ActiveSheet.ListObjects(strListobjectname).ListColumns.Count
    For i = 0 To intSpaltenzahl - 1
        ReDim Preserve sngSpaltenbreite(i)
            sngSpaltenbreite(i) =
ActiveSheet.ListObjects(strListobjectname).ListColumns(i) +

```

```

1).Range.ColumnWidth
Next i
With Me
    With .ListBox1
        .Clear
        .ListStyle = fmListStylePlain
        .ColumnCount = intSpaltenzahl
        .ColumnHeads = True
        For i = 0 To intSpaltenzahl - 1
            varSpaltenbreiten = varSpaltenbreiten &
                CStr(sngSpaltenbreite(i) / 5.3 & " cm;")
        Next i
        .FontSize = 10.5
        .ColumnWidths = varSpaltenbreiten
        Call RowSourceEinstellen
    End With
End With
End Sub

Sub RowSourceEinstellen()
    With ListBox1
        .RowSource =
            ActiveSheet.ListObjects(strListObjectName).Range.Address
    If
        ActiveSheet.ListObjects(strListObjectName).Range.Rows.Count > 1 Then

```



```

        .RowSource =
ActiveSheet.ListObjects(strListobjectname).Range.Offset(1, 0).Resize(
-
ActiveSheet.ListObjects(strListobjectname).Range.Rows.Count -
1).Address(External:=True)
        End If
        End With
    End Sub

Private Sub CommandButton1_Click()
    Dim Listzeile As ListRow, Bereich As Range, i%, j%, tb As
MSForms.Control
    Set Listzeile =
ActiveSheet.ListObjects(strListobjectname).ListRows.Add
    Set Bereich = Activ
eSheet.ListObjects(strListobjectname).ListRows(Listzeile.Index).Range
    i = 1: j = Listzeile.Index
    For Each tb In Me.Controls
        If TypeName(tb) = "TextBox" Then
            Bereich(i) = tb.Text
            i = i + 1
        End If
    End If
    If i >
ActiveSheet.ListObjects(strListobjectname).ListColumns.Count Then

```

```

Exit For
Next tb
Call RowSourceEinstellen
ListBox1.Selected(j - 1) = True
For Each tb In Me.Controls
    If TypeName(tb) = "TextBox" Then
        tb.Text = ""
    End If
Next tb
End Sub

Private Sub CommandButton2_Click()
    Dim i%, j%, Bereich As Range, varBereich() As Variant, tb As
MSForms.Control
    i = 1
    If ListBox1.ListIndex = -1 Then ListBox1.Selected(0) = True
    j = ListBox1.ListIndex
    On Error Resume Next
    Set Bereich = Acti
veSheet.ListObjects(strListobjectname).ListRows(Me.ListBox1.ListIndex
+ 1).Range
    For Each tb In Me.Controls
        If TypeName(tb) = "TextBox" Then
            ReDim Preserve varBereich(i)
            varBereich(i) = tb.Text

```

```

i = i + 1
End If
If i >
ActiveSheet.ListObjects(strListObjectName).ListColumns.Count Then
Exit For
Next tb
For i = 1 To UBound(varBereich)
Bereich(i) = varBereich(i)
Next i
Call RowSourceEinstellen
ListBox1.Selected(j) = True
For Each tb In Me.Controls
If TypeName(tb) = "TextBox" Then
tb.Text = ""
End If
Next tb
End Sub

Private Sub CommandButton3_Click()
Dim i&, tb As MSForms.Control
i = ListBox1.ListIndex
On Error Resume Next
Acti
veSheet.ListObjects(strListObjectName).ListRows(Me.ListBox1.ListIndex
+ 1).Delete

```

```

Call RowSourceEinstellen
On Error Resume Next
ListBox1.Selected(i - 1) = True
For Each tb In Me.Controls
    If TypeName(tb) = "TextBox" Then
        tb.Text = ""
    End If
Next tb
End Sub

Private Sub ListBox1_click()
    Dim i%, Bereich As Range, tb As MSForms.Control
    i = 1
On Error Resume Next
    Set Bereich = Acti
veSheet.ListObjects(strListobjectname).ListRows(Me.ListBox1.ListIndex
+ 1).Range
For Each tb In Me.Controls
    If TypeName(tb) = "TextBox" Then
        tb.Text = Bereich(i)
        i = i + 1
    End If
    If i >
ActiveSheet.ListObjects(strListobjectname).ListColumns.Count Then
Exit For

```

```
Next tb
End Sub

Private Sub UserForm_Initialize()
    strListObjectName = Selection.ListObject.Name
    Call ListenfeldFüllen
End Sub
```

24.6. Tabellenlistenzeilen scrollen

Erzeugen Sie ein Drehfeld und erzeugen per Makro zum Testen eine Tabellenliste. Die letztgenannten Makros kopieren Sie in das Codemodul des verwendeten Arbeitsblatts.

Standardmodul

```
Sub SpinbuttonEinfügen()
    Dim cb As OLEObject
```

```

Set cb =
ActiveSheet.OLEObjects.Add(ClassType:="Forms.SpinButton.1",
Link:=False, _
DisplayAsIcon:=False, Left:=413.25, Top:=86.25, Width:=28.5,
Height:=33)
End Sub

Sub CreateTable()
[a1] = "Produkt": [b1] = "Verkäufer": [c1] = "Verkaufsmenge"
[a2] = "Navigation": [b2] = "Schröder": [c2] = 1
[a3] = "Handy": [b3] = "Schmied": [c3] = 10
[a4] = "Navigation": [b4] = "Müller": [c4] = 20
[a5] = "Navigation": [b5] = "Schmied": [c5] = 30
[a6] = "Handy": [b6] = "Müller": [c6] = 40
[a7] = "iPod": [b7] = "Schmied": [c7] = 50
[a8] = "Navigation": [b8] = "Schröder": [c8] = 60
[a9] = "Handy": [b9] = "Becker": [c9] = 70
[a10] = "iPod": [b10] = "Müller": [c10] = 80
On Error Resume Next
ActiveSheet.ListObjects.Add(xlSrcRange, Range("$a$1:$c$10"), ,
xlYes).Name = _
"Table1"
ActiveSheet.ListObjects("Table1").TableStyle = "TableStyleLight2"
End Sub

```

Codemodul des Arbeitsblatts mit der Tabellenliste

```
Private lo As ListObject, lr As ListRow
Private lngSpinbutton1Max, lngSpinSelected&

Private Sub Worksheet_Activate ()
    Call Werte
End Sub

Private Sub Worksheet_SelectionChange (ByVal Target As Range)
    Call Werte
End Sub

Private Sub SpinButton1_SpinUp ()
    Call swap
End Sub

Private Sub SpinButton1_SpinDown ()
    Call swap
End Sub

Private Sub Werte ()
    If Not Intersect (ActiveCell, ListObjects(1).DataBodyRange) Is
Nothing Then
```

```

SpinButton1.Max =
ActiveSheet.ListObjects(Selection.ListObject.Name).ListRows.Count
SpinButton1.Min = 1
lngSpinbutton1Max = SpinButton1.Max
Set lo = ActiveSheet.ListObjects("Table1")
For Each lr In lo.ListRows
    If Not Intersect(lr.Range, ActiveCell) Is Nothing Then
        SpinButton1.Value = lngSpinbutton1Max + 1 - lr.Index
        lngSpinSelected = lr.Index
        Exit For
    End If
Next lr
End If
End Sub

Private Sub swap()
    Dim lngSpinNeu&
    Dim ZeileNeu As Range, ZeileAlt As Range
    Dim varZeileNeu As Variant, varZeileAlt As Variant, varMerkZeile

On Error Resume Next
If Not Intersect(ActiveCell,
ListObjects(Selection.ListObject.Name).DataBodyRange) Is Nothing Then
    lngSpinNeu = SpinButton1.Max + 1 - SpinButton1.Value

```



```
If lngSpinNeu <> lngSpinSelected Then  
    Set ZeileNeu = ActiveSheet.ListObjects(Selection.ListObject.Name).ListRows(lngSpinNeu).Range  
    Set ZeileAlt = ActiveSheet.ListObjects(Selection.ListObject.Name).ListRows(lngSpinSelected).Range  
    varZeileNeu = ZeileNeu  
    varZeileAlt = ZeileAlt  
  
    varMerkZeile = varZeileNeu  
    varZeileNeu = varZeileAlt  
    varZeileAlt = varMerkZeile  
  
    ZeileAlt = varZeileAlt  
    ZeileNeu = varZeileNeu  
    lngSpinSelected = lngSpinNeu  
    ActiveSheet.ListObjects(Selection.ListObject.Name).ListRows(lngSpinSelected).Range.Select  
  
End If  
End If  
End Sub
```

24.7. Exceldaten in XML-Dokument exportieren

Soweit mit Ihrer Office-Version XML mitgeliefert wurde, setzen Sie einen Verweis auf Microsoft XML. Dieses Makro verwendet die Version 6.0. Bei Version 5.0 verwenden Sie die Variablendeklaration `Domdocument50`.

```
Sub Excel_XML()
    Dim xml As New MSXML2.Domdocument60
    Dim xmlKnoten As MSXML2.IXMLDOMElement
    Dim xmlUnterknoten As MSXML2.IXMLDOMElement
    Dim Zelle As Range, strWert$, strNeu$, i%
    Cells.Clear
    [a1] = "Produkt": [b1] = "Verkäufer": [c1] = "Verkaufsmenge"
    [a2] = "Navigation": [b2] = "Schröder": [c2] = 1
    [a3] = "Handy": [b3] = "Schmied": [c3] = 10
    [a4] = "Navigation": [b4] = "Müller": [c4] = 20
    [a5] = "Navigation": [b5] = "Schmied": [c5] = 30
    [a6] = "Handy": [b6] = "Müller": [c6] = 40
    [a7] = "iPod": [b7] = "Schmied": [c7] = 50
    [a8] = "Navigation": [b8] = "Schröder": [c8] = 60
    [a9] = "Handy": [b9] = "Becker": [c9] = 70
    [a10] = "iPod": [b10] = "Müller": [c10] = 80
End Sub
```

```

xml.LoadXML "<?xml version="1.0" " & "
encoding="ISO-8859-1"?"><meineXMLListe/>"
For Each Row In [a2:c10].Rows
    Set xmlKnoten = xml.createElement("Knoten")
    For Each Zelle In [a1:c1].Columns
        Zelle.Value = Replace(Zelle.Value, "ä", "ae")
        Zelle.Value = Replace(Zelle.Value, "Ä", "Ae")
        Zelle.Value = Replace(Zelle.Value, "ö", "oe")
        Zelle.Value = Replace(Zelle.Value, "Ö", "Oe")
        Zelle.Value = Replace(Zelle.Value, "ü", "ue")
        Zelle.Value = Replace(Zelle.Value, "Ü", "Ue")
    For i = 1 To Len(Zelle.Value)
        If Mid(Zelle.Value, i, 1) Like "[a-z]" Or
Mid(Zelle.Value, i, 1) Like "[A-Z]" Or _
Mid(Zelle.Value, i, 1) Like "[0-9]" Or
Mid(Zelle.Value, i, 1) Like "_" Then _
    strNeu = strNeu & Mid(Zelle.Value, i, 1)
    Next i
    Set xmlUnterknoten = xml.createElement(strNeu)
    xmlKnoten.appendChild(xmlUnterknoten).Text =
Cells(Row.Row, Zelle.Column).Value
    strNeu = ""
Next Zelle
xml.DocumentElement.appendChild xmlKnoten
Next Row

```

```

xml.Save Environ("tmp") & "\meineXMLDatei.xml"
Set xml = Nothing: Set xmlKnoten = Nothing: Set xmlUnterknoten =
Nothing
End Sub

```

24.8. XML-Daten in Excelblatt importieren

Erzeugen Sie mit dem ersten Makro die Schemadefinition. Der Import erfolgt dann mit dem zweiten Makro, das die Schema-Definition verwendet.

```

Sub Create_XSD()
  Dim strMyXml As String, meinMap As XmlMap
  Dim strMeinSchema$
  strMyXml = "<meineXMLListe>" & _
    "<Knoten>" & _
    "<Produkt>Text</Produkt>" & _
    "<Verkaeufuer>Text</Verkaeufuer>" & _
    "<Verkaufsmenge>999</Verkaufsmenge>" & _
    "</Knoten>" & _

```

```

"<Knoten></Knoten>" & _
"</meineXMLListe>"
Application.DisplayAlerts = False
Set meinMap = ThisWorkbook.XmlMaps.Add(strMyXml)
Application.DisplayAlerts = True
strMeinSchema = meinMap.Schemas(1).xml
Open ThisWorkbook.Path & "\strMeinSchema.xsd" For Output As #1
Print #1, strMeinSchema
Close #1
End Sub

Sub CreateXMLList ()
  Dim Map1 As XmlMap
  Dim objList As ListObject
  Dim objColumn As ListColumn
  Dim i%
  If Dir(ThisWorkbook.Path & "\strMeinSchema.xsd") = "" Then Exit
Sub
  Set Map1 = ThisWorkbook.XmlMaps.Add(ThisWorkbook.Path &
  "\strMeinSchema.xsd")
  On Error Resume Next
  ActiveSheet.ListObjects(1).Delete
  Application.DisplayAlerts = False
  ActiveSheet.Range("A1").Select
  Set objList = ActiveSheet.ListObjects.Add

```

```

objList.ListColumns(1).XPath.SetValue Map1,
"/meineXMLListe/Knoten/Produkt"
Set objColumn = objList.ListColumns.Add
objColumn.XPath.SetValue Map1, "/meineXMLListe/Knoten/Verkaeuer"

Set objColumn = objList.ListColumns.Add
objColumn.XPath.SetValue Map1,
"/meineXMLListe/Knoten/Verkaufsmenge"
objList.ListColumns(1).Name = "Produkt"
objList.ListColumns(2).Name = "Verkäufer"
objList.ListColumns(3).Name = "Verkaufsmenge"
Columns.AutoFit
Application.DisplayAlerts = False
Map1.Import (Environ("tmp") & "\meineXMLDatei.xml")
End Sub

```

oder:

Standardmodul

```

Public Sub GetOverwrite()
Dim clsOverwrite As New Klasse1
Cells.Clear

```

```
On Error Resume Next
clsOverwrite.GetXMLData
End Sub
```

Klassenmodul, Name: Klasse1

```
Public Function GetXMLData() As Variant
Dim strXmlQuelldatei$
Dim XmlImportResult As XmlImportResult

strXmlQuelldatei = Environ("tmp") & "\meineXMLDatei.xml"

If Dir(strXmlQuelldatei) = vbNullString Then MsgBox "Die
Quelldatei wurde nicht gefunden"

XmlImportResult = ActiveWorkbook.XmlImport(strXmlQuelldatei,
Nothing, _
True, ActiveCell)
If XmlImportResult = xlXmlImportSuccess Then MsgBox "XML
Datenimport komplett"
End Function
```

24.9. Exceldaten in Access-Datenbank exportieren

```
Sub neueDatenbankErzeugen ()
    Dim cat As New ADOX.Catalog
    Dim tbl As New ADOX.Table
    Dim strPfad$
    strPfad = Environ("localAPPDATA") &
"\microsoft\office\pivotTabelle.accdb"
    If Dir(strPfad) = "" Then _
        cat.Create "Provider = microsoft.ace.oledb.12.0; data
source=" & strPfad
    With tbl
        .ParentCatalog = cat
        .Name = "Früchteverkauf"
        With .Columns
            .Append "Frucht", adVarChar, 60
            .Append "Monat", adVarChar, 10
            .Append "Menge", adInteger
        End With
        .Columns("Menge").Properties("Nullable") = True
    End With
    cat.Tables.Append tbl
Set tbl = Nothing
```



```

Set cat = Nothing
End Sub

Sub DatenHinzufügenADO()
Dim conn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim Row As Range, Column As Range
Dim strPfad$
strPfad = Environ("localAPPDATA") &
"\microsoft\office\pivotTabelle.accdb"
If Dir(strPfad) = "" Then Exit Sub
With ActiveSheet
.Cells.Clear
.[a1] = "Frucht": .[B1] = "Jan.": .[C1] = "Feb.": .[D1]
= "Mär."
.[A2] = "Äpfel": .[B2] = 5: .[C2] = 3: .[D2]
= 4
.[a3] = "Orangen": .[B3] = 4: .[D3]
= 5
.[A4] = "Birnen": .[B4] = 2: .[C4] = 3: .[D4]
= 5
conn.Open "Provider=Microsoft.ace.OLEDB.12.0;" & _
"Data Source=" & strPfad
End With
With rs

```

```

.Open "Früchteverkauf", conn, adOpenKeyset, adLockOptimistic

For Each Row In ActiveSheet.[2:4].Rows
    For Each Column In ActiveSheet.[b:d].Columns
        .AddNew
        !Frucht = ActiveSheet.Cells(Row.Row, 1)
        !Monat = ActiveSheet.Cells(1, Column.Column)
        !Menge = ActiveSheet.Cells(Row.Row, Column.Column)
        .Update
    Next Column
Next Row
.Close
End With
Set rs = Nothing: Set conn = Nothing
End Sub

```

24.10. Pivottable aus Accessdatenbank erstellen

```

Sub CreatePivotTableADO()
    Dim PivotC As PivotCache
    Dim PivotT As PivotTable

```

```

Dim strSQL$
Dim conn As New ADODB.Connection
Dim rs As New ADODB.Recordset
conn.Open "Provider=Microsoft.ace.OLEDB.12.0;" & "Data Source=" &
-
    Environ("localAPPDATA") &
    "\microsoft\office\pivotTabelle.accdb"
rs.Open "Früchteverkauf", conn, adOpenKeyset, adLockOptimistic
If rs.RecordCount = 0 Then MsgBox ("Keine Datensätze gefunden!"),
vbCritical
ActiveWindow.DisplayGridlines = False
Set PivotC =
ActiveWorkbook.PivotCaches.Create(SourceType:=xlExternal)
Set PivotC.Recordset = rs
Worksheets.Add Before:=Sheets(1)
Set PivotT = ActiveSheet.PivotTables.Add(PivotCache:=PivotC,
-
    TableDestination:=ActiveSheet.Range("a3"))
With PivotT
    .NullString = "0"
    .AddFields RowFields:="Frucht", ColumnFields:="Monat"
    .PivotFields("Menge").Orientation = xlDataField
End With
Set rs = Nothing
Set conn = Nothing

```

```
Set PivotT = Nothing
Set PivotC = Nothing
End Sub
```

24.11. Formula Array

Wechseln im Menü Excel-Option/Formeln zum S1Z1-Bezugsstil.

Das Makro erzeugt für einen Test eine Tabellenliste. Geben Sie in die Inputboxen einen Verkäufernamen und einen Produktnamen ein. Als Ergebnis erhalten Sie zunächst eine Information, welche Gesamtmenge des Produkts der Verkäufer insgesamt veräußert hat. Darüber hinaus wird Auskunft gegeben, um wieviele Tabellenpositionen es geht. Geben Sie für einen Test den Verkäufernamen Schröder und den Produktnamen Navigation ein!

```
Sub testMich ()
Dim strProdukt$
Dim strVerkäufer
```

```

Dim strSpalte1
Dim strSpalte2
Dim strSpalte3
Dim Bereich1 As Range
Dim Bereich2 As Range
Dim Zelle As Range
Dim bool As Boolean
With ActiveSheet
    .Cells.Clear
    .ListObjects.Add(xlSrcRange, Range("$a$1:$c$10"), ,
xlYes).Name = "Table1"
    .ListObjects("Table1").TableStyle = "TableStyleLight2"
    .[a1] = "Produkt": .[b1] = "Verkäufer": .[c1] =
"Verkaufsmenge"
    .[a2] = "Navigation": .[b2] = "Schröder": .[c2] = 1
    .[a3] = "Handy": .[b3] = "Schmied": .[c3] = 10
    .[a4] = "Navigation": .[b4] = "Müller": .[c4] = 20
    .[a5] = "Navigation": .[b5] = "Schmied": .[c5] = 30
    .[a6] = "Handy": .[b6] = "Müller": .[c6] = 40
    .[a7] = "iPod": .[b7] = "Schmied": .[c7] = 50
    .[a8] = "Navigation": .[b8] = "Schröder": .[c8] = 60
    .[a9] = "Handy": .[b9] = "Becker": .[c9] = 70
    .[a10] = "iPod": .[b10] = "Müller": .[c10] = 80
strSpalte1 = Acti

```

```

veSheet.ListObjects("Table1").DataBodyRange.Columns(1).Address (False,
False)
    strSpalte2 = Acti
veSheet.ListObjects("Table1").DataBodyRange.Columns(2).Address (False,
False)
    strSpalte3 = Acti
veSheet.ListObjects("Table1").DataBodyRange.Columns(3).Address (False,
False)
    Set Bereich1 = Range(strSpalte1)
    Set Bereich2 = Range(strSpalte2)
    strProdukt = InputBox("Gebe das Produkt ein!")
    If strProdukt = " " Then Exit Sub
    For Each Zelle In Bereich1
        If Zelle.Value = strProdukt Then bool = True
    Next Zelle
    If bool = False Then
        MsgBox "Der eingegebene Produktname existiert nicht oder
ist falsch", vbInformation
    Exit Sub
End If
bool = False
strVerkäufer = InputBox("Gebe den Verkäufer ein!")
If strVerkäufer = " " Then Exit Sub
For Each Zelle In Bereich2
    If Zelle.Value = strVerkäufer Then bool = True

```

```

Next Zelle
If bool = False Then
    MsgBox "Der eingegebene Verkäufername existiert nicht
oder ist falsch", vbInformation
Exit Sub
End If
    .[e9] = "Gesamte Verkaufsmenge " & strProdukt & " durch
Verkäufer " & strVerkäufer
    .[e10].FormulaArray = "=SUM((" & strSpalte1 & "= "" " &
strProdukt & """)*( " & strSpalte2 & """" & strVerkäufer & """)*( " &
strSpalte3 & ""))"
    .[e12] = "Anzahl der Verkaufspostitionen des Produkts " &
strProdukt & " duch den Verkäufer " & strVerkäufer 'logischen UND
letztlich aber ANZAHL der Zeilen mit Navigation von Schröder
---works---
    .[e13].FormulaArray = "=SUM((" & strSpalte1 & "= "" " &
strProdukt & """)*( " & strSpalte2 & " = """" & strVerkäufer & """"))"
    End With
End Sub

```

24.12. Bedingte Formatierung

Dieses Beispiel erzeugt anhand einer Beispieltabelle mit bedingter Formatierung Richtungspfeile, die abhängig vom Trend in eine bestimmte Richtung zeigen.

```
Sub SetConditionalFormatting()  
    Dim cfIconSet As IconSetCondition: Dim Bool As Boolean  
    For Each Worksheet In ThisWorkbook.Worksheets  
        If Worksheet.Name = "Bedingte Formatierung" Then Bool = True  
    Next Worksheet  
    If Bool = False Then  
        Worksheets.Add(After:=Worksheets(Worksheets.Count)).Name = "Bedingte  
Formatierung"  
        With Sheets("Bedingte Formatierung")  
            .Cells.Clear  
            .Range("C1").Value = -0.01: .Range("C6").Value = 0  
            .Range("C2").Value = 0.005: .Range("C7").Value = 0  
            .Range("C3").Value = -0.02: .Range("C8").Value = 0.005  
            .Range("C4").Value = -0.02: .Range("C9").Value = -0.02  
            .Range("C5").Value = 0.005: .Range("C10").Value = 0.005  
            .Range("C1", "C10").NumberFormat = " 0.00 ;[Red] - 0.00 "
```



```
Set cfIconSet = .Range("C1",  
"C10").FormatConditions.AddIconSetCondition  
.Range("C1", "C10").FormatConditions(1).SetFirstPriority  
End With  
cfIconSet.IconSet = ActiveWorkbook.IconSets(xl3Arrows)  
With cfIconSet.IconCriteria(2)  
.Type = xlConditionValueNumber  
.Value = 0  
.Operator = 7  
End With  
With cfIconSet.IconCriteria(3)  
.Type = xlConditionValueNumber  
.Value = 0.0001  
.Operator = 7  
End With  
Set cfIconSet = Nothing  
End Sub
```

24.13. Zellengroße Diagramme in Arbeitsblatt einfügen

Dieses Beispiel erzeugt anhand einiger Testdaten zellengroße Säulendiagramme.

```
Sub addTinyCharts ()  
  Dim Bereich As Range  
  Dim i As Integer  
  With ActiveSheet  
    Set Bereich = .[b2:m4]  
    For i = .ChartObjects.Count To 1 Step -1  
      .ChartObjects(i).Delete  
    Next i  
    .[a1] = "Frucht":      .[B1] = "Jan.":      .[C1] =  
    "Feb.":  .[D1] = "Mär.":  .[E1] = "Apr.":      .[f1] = "Mai":  .[g1]  
    = "Jun.":  .[h1] = "Jul.":  .[i1] = "Aug.":  .[j1] = "Sep.":  .[k1] =  
    "Okt.":  .[l1] = "Nov.":  .[m1] = "Dez.":  .[n1] = "Gesamt"  
    .[A2] = "Ananas":      .[B2] = 5:          .[C2] = 3:  
    .[D2] = 4:              .[e2] = 4:          .[f2] = 4:          .[g2] = 4:  
    .[h2] = 4:              .[i2] = 4:          .[j2] = 4:          .[k2] = 4:
```

```

.[l2] = 4:      .[m2] = 4
                .[a3] = "Kiwī":      .[B3] = 45:      .[C3] = 78:
                .[D3] = 78:      .[e3] = 78:      .[f3] = 98:      .[g3] = 88:
                [h3] = 4:      .[i3] = 4:      .[j3] = 4:      .[k3] = 8:
                .[l3] = 69:      .[m3] = 96
                .[A4] = "Papaya":      .[B4] = 54:      .[C4] = 27:
                .[D4] = 33:      .[e4] = 82:      .[f4] = 4:      .[g4] = 4:
                .[h4] = 4:      .[i4] = 4:      .[j4] = 51:      .[k4] = 10:
                .[l4] = 4:      .[m4] = 10
                .[n2].Formula = "=Sum(b2:m2) "
                .[n2].AutoFill .Range("n2:n4"), xlFillDefault
End With

```

For Each Row In Bereich.Rows

```

ActiveSheet.Shapes.AddChart.Select
With ActiveChart
    .ChartType = xlColumnClustered
    .SetSourceData Source:=Row
    .HasLegend = False
    .HasTitle = False
    .Axes(xlCategory, xlPrimary).HasTitle = False
    .Axes(xlValue, xlPrimary).HasTitle = False
    .HasAxis(xlCategory, xlPrimary) = False
    .HasAxis(xlValue, xlPrimary) = False
    .Axes(xlValue).MajorGridlines.Delete
    .Axes(xlValue).MinorGridlines.Delete

```

```
.Axes(xlCategory).MajorGridlines.Delete
.Axes(xlCategory).MinorGridlines.Delete
.SeriesCollection(1).Interior.ColorIndex = 37
.SeriesCollection(1).Border.ColorIndex = 25
.Parent.Top = ActiveSheet.Cells(Row.Row, 15).Top + 1
.Parent.Left = ActiveSheet.Cells(Row.Row, 15).Left + 1
.Parent.Height = ActiveSheet.Cells(Row.Row, 15).Height -

.Parent.Width = ActiveSheet.Cells(Row.Row, 15).Width - 2
.Parent.Border.ColorIndex = xlNone
.PlotArea.Top = 0
.PlotArea.Left = 0
.PlotArea.Height = .Parent.Height
.PlotArea.Width = .Parent.Width
.ChartGroups(1).GapWidth = 50
```

End With

Next Row

End Sub

24.14. Datensatzkollektion anlegen

Datensätze lassen sich in Datenfelder (Arrays) kopieren. Allerdings ist dann eine flexible Handhabung der Datensätze kaum möglich. Insofern besteht die bessere Alternative, mit Klassenmodulen zu arbeiten und Kollektionen anzulegen.

Beachte: Kopieren Sie die letzten beiden Makros nicht in ein Standard- sondern in ein Klassenmodul. Benennen Sie die im Beispiel genannten Klassenmodule jeweils im Eigenschaftfenster mit `clsKontakt` und `clsKontakte`.

Folgendes Makro wäre möglich:

```
Option Base 1
Type Anwenderkontakttdaten
  LfdNr As String
  Nachname As String * 25
  HerrFrau As Boolean
  Fon As String * 25
End Type
```

```

Sub ArrayFüllen ()
    Dim PersAngaben () As Anwenderkontaktdaten
    Dim i As Integer
    [a1] = "Lfdnr": [b1] = "Nachname": [c1] = "HerrFrau": [d1] =
        "Fon"
    [a2] = "1": [b2] = "Becker": [c2] = "False": [d2] =
        "123"
    [a3] = "2": [b3] = "Becher": [c3] = "True": [d3] =
        "234"
    [a4] = "3": [b4] = "Bäcker": [c4] = "0": [d4] =
        "456"
    For i = 2 To ActiveSheet.UsedRange.Rows.Count
        ReDim Preserve PersAngaben(i)
        PersAngaben(i).LfdNr = Cells(i, 1)
        PersAngaben(i).Nachname = Cells(i, 2)
        PersAngaben(i).HerrFrau = Cells(i, 3)
        PersAngaben(i).Fon = Cells(i, 4)
    Next i
    MsgBox "Funktionstest: Im ersten Datensatz " & _
        "steht der Wert " & PersAngaben(2).LfdNr, vbInformation
End Sub

```

Besser ist folgende Variante:

Standardmodul:

```
option explicit
Sub TestKontakteClass()
    Dim Kontakt As clsKontakt
    Dim Kontakte As New clsKontakte
    Dim i As Integer
    [a1] = "Lfdnr": [b1] = "Nachname": [c1] = "HerrFrau": [d1] =
        "Fon"
    [a2] = "1": [b2] = "Becker": [c2] = "False": [d2] =
        "123"
    [a3] = "2": [b3] = "Becher": [c3] = "True": [d3] =
        "234"
    [a4] = "3": [b4] = "Bäcker": [c4] = "0": [d4] =
        "456"
    For i = 2 To ActiveSheet.[a1].CurrentRegion.Rows.Count
        Set Kontakt = New clsKontakt
        Kontakt.LfdNr = ActiveSheet.Cells(i, 1)
        Kontakt.Nachname = ActiveSheet.Cells(i, 2)
        Kontakt.HerrFrau = CBool(ActiveSheet.Cells(i, 3))
        Kontakt.Fon = ActiveSheet.Cells(i, 4)
        Kontakte.Add Kontakt
    Next i
    Kontakte.Remove 2
```

```
MsgBox "Nachdem der Kontakt Nr. 2 gelöscht wurde," & _  
"beträgt die Anzahl der Kontakte " & Kontakte.Count & "." & vbCrLf  
& _  
"Jetzt hat der zweite Kontakt die laufende Nummer " &  
Kontakte.Item(2).LfdNr & "."  
Set Kontakte = Nothing  
End Sub
```

Klassenmodul, Name: "clsKontakt"

```
Option Explicit  
Dim pLfdNr As String  
Dim pNachname As String  
Dim pHerrFrau As Boolean  
Dim pFon As String  
Public KontaktID As String  
  
Public Property Get LfdNr() As String  
LfdNr = pLfdNr  
End Property  
  
Public Property Let LfdNr(strLfdNr As String)  
pLfdNr = strLfdNr  
End Property
```



```
Public Property Get Nachname() As String
    Nachname = pNachname
End Property

Public Property Let Nachname(strNachname As String)
    pNachname = strNachname
End Property

Public Property Get HerrFrau() As Boolean
    HerrFrau = pHerrFrau
End Property

Public Property Let HerrFrau(boolHerrFrau As Boolean)
    pHerrFrau = boolHerrFrau
End Property

Public Property Get Fon() As String
    Fon = pFon
End Property

Public Property Let Fon(strFon As String)
    pFon = strFon
End Property
```

Klassenmodul, Name: "clsKontakte"

```
Option Explicit
Private KontakteP As Collection

Public Property Get Count() As Long
    Count = KontakteP.Count
End Property

Public Function Item(Index As Variant) As clsKontakt
    Set Item = KontakteP(Index)
End Function

Public Sub Add(Kontakt As clsKontakt)
    On Error Goto AddError
    KontakteP.Add Kontakt
Exit Sub
AddError:
    Err.Raise Number:=vbObjectError + 514, Source:="clsKontakte.Add",
-
    Description:="Unable to Add clsKontakt object to the
collection"
End Sub
```

```
Public Sub Remove(ByVal Index As Integer)  
    On Error GoTo RemoveError  
    KontakteP.Remove Index  
    Exit Sub  
RemoveError:  
    Err.Raise Number:=vbObjectError + 515,  
    Source:="clsKontakte.Remove", _  
    Description:="Das clsCell object kann nicht von der  
    Kollektion gelöscht werden!"  
End Sub  
  
Private Sub Class_Initialize()  
    Set KontakteP = New Collection  
End Sub  
  
Private Sub Class_Terminate()  
    Set KontakteP = Nothing  
End Sub
```


25. Excel-Links

25.1. Deutschsprachige Links

- Herbers Excel Server¹ – mit Forum², Archiv³ und Excel-FAQ⁴
- Excel-Inside von Alois Eckl – Programmierung, VBA etc.⁵

1 <http://www.herber.de>

2 <http://xlforum.herber.de>

3 <http://xlarchiv.herber.de>

4 <http://xlfaq.herber.de>

5 <http://www.excel-inside.de/>

- Monika Weber⁶
- Bernd Held⁷
- Marcus Schmidt⁸
- Workstream.de – Tipps und Forum zu Excel und VBA⁹
- Thomas Risi¹⁰
- Computerwissen.de – Tipps zu Excel¹¹
- Online-Kurse¹²
- Broukals VBA-Kurs als PDF-Datei¹³
- Online-Kurs von Alexander Koch¹⁴

6 <http://www.jumper.ch/>

7 <http://held-office.de/>

8 <http://www.schmittis-page.de/>

9 <http://www.workstream.de/>

10 <http://rtsoftwaredevelopment.de/>

11 <http://www.computerwissen-inside.de/>

12 <http://www.excel-training.de/>

13 <http://www.joanneum.ac.at/services/vbaexcel/>

14 <http://lawww.de/Library/ExcelBasics/index.shtml>

- Die Excel-Formel-Seite¹⁵
- J. Häußler – Excel für Chemiker und Lehrer¹⁶
- Excel-Center Bernd Busko¹⁷
- Thomas Igel¹⁸
- I. Dietrich¹⁹
- Alexander Fuchs – Mathematik mit Excel²⁰
- EXCEL für Mathe Dummies²¹
- Marcus Rose – Excel-Treff²²
- Das Excel-2000-Lexikon²³

15 <http://www.excelformeln.de/>

16 <http://www.excelchem.de>

17 <http://www.excel-center.de/index.php>

18 <http://www.igel.net.de>

19 <http://www.i-dietrich.de/>

20 <http://www.geocities.com/ResearchTriangle/Forum/9137/>

21 <http://www.excelmexel.de>

22 <http://ms-excel.eu/>

23 <http://www.kmbuss.de/>

- Excel and more²⁴
- Sämtliche Limitationen und Spezifikationen²⁵
- Kostenlose Tutorials Helmut Mittelbach²⁶
- Excel-Tuning, Excel-AddIn mit 300 Makros und allen Shortcuts²⁷
- Yexcel – Das Excel-Portal²⁸
- Die Excel-Wühlkiste²⁹

25.2. Englischsprachige Links

- MrExcel³⁰

24 <http://www.xlam.ch/>

25 <http://www.xlam.ch/xlimits/index.htm>

26 <http://www.excelmexel.de/fkurse.htm/>

27 <http://www.excel-tuning.de/>

28 <http://www.yexcel.de/>

29 <http://www.scheidgen.de/>

30 <http://www.mrexcel.com/>

- David McRitchie³¹
- Tools für Excel³²
- Passwort-Cracker³³
- Chip Pearson³⁴
- Beyond Technology³⁵
- John Walkenbach³⁶
- Rob Bovey³⁷
- Dave Steppan³⁸

31 <http://www.geocities.com/davemcritchie/>

32 <http://www.bookcase.com/library/software/win3x.apps.excel.html>

33 <http://www.lostpassword.com/excel.htm>

34 <http://www.cpearson.com/>

35 <http://www.beyondtechnology.com/>

36 <http://www.j-walk.com/>

37 <http://www.appspro.com/>

38 <http://www.geocities.com/SiliconValley/Network/1030/ExcelTop.html>

- John F. Lacher³⁹
- Bob Umlas – Rechnen mit Array-Formeln⁴⁰
- Fred Cummings⁴¹
- Igor Kolupaev⁴²
- Ole Erlandsen⁴³
- VirtualHelpDesk⁴⁴
- Alan Barasch⁴⁵
- Ture Magnusson⁴⁶

39 <http://www.lacher.com/>

40 <http://www.emailoffice.com/excel/arrays-bobumlas.html>

41 <http://www.netspace.net.au/~fcfhsp/xlhome.htm>

42 <http://www.geocities.com/SiliconValley/Lab/5586/>

43 <http://www.erlandsendata.no/english/>

44 <http://keptin.net/virtualhelpdesk/excel/>

45 <http://xl.barasch.com/>

46 <http://www.turedata.se/excel/>

26. Autoren

Edits	User
11	Albin ¹
1	Bukk ²
1	BurkhardS ³
1	Calle Cool ⁴

¹ <http://de.wikibooks.org/w/index.php?title=Benutzer:Albin>

² <http://de.wikibooks.org/w/index.php?title=Benutzer:Bukk>

³ <http://de.wikibooks.org/w/index.php?title=Benutzer:BurkhardS>

⁴ http://de.wikibooks.org/w/index.php?title=Benutzer:Calle_Cool

40 Dirk Huenniger⁵

1 Dr. scEye⁶

2 Eri474⁷

1 Felgentraeger⁸

6 Foso⁹

53 Geitost¹⁰

27 Hans W. Herber¹¹

5 [http://de.wikibooks.org/w/index.php?title=Benutzer:](http://de.wikibooks.org/w/index.php?title=Benutzer:Dirk_Huenniger)

Dirk_Huenniger

6 [http://de.wikibooks.org/w/index.php?title=Benutzer:](http://de.wikibooks.org/w/index.php?title=Benutzer:Dr._scEye)

Dr._scEye

7 <http://de.wikibooks.org/w/index.php?title=Benutzer:Eri474>

8 [http://de.wikibooks.org/w/index.php?title=Benutzer:](http://de.wikibooks.org/w/index.php?title=Benutzer:Felgentraeger)

Felgentraeger

9 <http://de.wikibooks.org/w/index.php?title=Benutzer:Foso>

10 <http://de.wikibooks.org/w/index.php?title=Benutzer:Geitost>

11 [http://de.wikibooks.org/w/index.php?title=Benutzer:](http://de.wikibooks.org/w/index.php?title=Benutzer:Hans_W._Herber)

Hans_W._Herber

- 1 Hardy42¹²
- 1 Heho¹³
- 1 Heuler06¹⁴
- 1 Imzadi¹⁵
- 1 Jonal¹⁶
- 11 Juetho¹⁷
- 3 Klaus Eifert¹⁸
- 1 Marose67¹⁹

-
- 12 <http://de.wikibooks.org/w/index.php?title=Benutzer:Hardy42>
 - 13 <http://de.wikibooks.org/w/index.php?title=Benutzer:Heho>
 - 14 <http://de.wikibooks.org/w/index.php?title=Benutzer:Heuler06>
 - 15 <http://de.wikibooks.org/w/index.php?title=Benutzer:Imzadi>
 - 16 <http://de.wikibooks.org/w/index.php?title=Benutzer:Jonal>
 - 17 <http://de.wikibooks.org/w/index.php?title=Benutzer:Juetho>
 - 18 http://de.wikibooks.org/w/index.php?title=Benutzer:Klaus_Eifert
 - 19 <http://de.wikibooks.org/w/index.php?title=Benutzer:Marose67>

- 2 MichaelFrey²⁰
- 1 Mijo S.²¹
- 5 Mjchael²²
- 3 NeuerNutzer2009²³
- 1 Ollio²⁴
- 1 Prog²⁵
- 25 Ralf Pfeifer²⁶

20 <http://de.wikibooks.org/w/index.php?title=Benutzer:MichaelFrey>

21 http://de.wikibooks.org/w/index.php?title=Benutzer:Mijo_S.

22 <http://de.wikibooks.org/w/index.php?title=Benutzer:Mjchael>

23 <http://de.wikibooks.org/w/index.php?title=Benutzer:NeuerNutzer2009>

24 <http://de.wikibooks.org/w/index.php?title=Benutzer:Ollio>

25 <http://de.wikibooks.org/w/index.php?title=Benutzer:Prog>

26 http://de.wikibooks.org/w/index.php?title=Benutzer:Ralf_Pfeifer

- 5 Ralle002²⁷
- 4 Rudolf73²⁸
- 1 Techni-Tom²⁹
- 8 ThePacker³⁰
- 2 TheUDF³¹
- 24 XIOtto³²

-
- 27 <http://de.wikibooks.org/w/index.php?title=Benutzer:Ralle002>
 - 28 <http://de.wikibooks.org/w/index.php?title=Benutzer:Rudolf73>
 - 29 <http://de.wikibooks.org/w/index.php?title=Benutzer:Techni-Tom>
 - 30 <http://de.wikibooks.org/w/index.php?title=Benutzer:ThePacker>
 - 31 <http://de.wikibooks.org/w/index.php?title=Benutzer:TheUDF>
 - 32 <http://de.wikibooks.org/w/index.php?title=Benutzer:XIOtto>

Abbildungsverzeichnis

- **GFDL: Gnu Free Documentation License.** <http://www.gnu.org/licenses/fdl.html>
- **cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License.**
<http://creativecommons.org/licenses/by-sa/3.0/>
- **cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License.**
<http://creativecommons.org/licenses/by-sa/2.5/>
- **cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License.**
<http://creativecommons.org/licenses/by-sa/2.0/>

- **cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License.** <http://creativecommons.org/licenses/by-sa/1.0/>
- **cc-by-2.0: Creative Commons Attribution 2.0 License.** <http://creativecommons.org/licenses/by/2.0/>
- **cc-by-2.0: Creative Commons Attribution 2.0 License.** <http://creativecommons.org/licenses/by/2.0/deed.en>
- **cc-by-2.5: Creative Commons Attribution 2.5 License.** <http://creativecommons.org/licenses/by/2.5/deed.en>
- **cc-by-3.0: Creative Commons Attribution 3.0 License.** <http://creativecommons.org/licenses/by/3.0/deed.en>
- **GPL: GNU General Public License.** <http://www.gnu.org/licenses/gpl-2.0.txt>
- **LGPL: GNU Lesser General Public License.** <http://www.gnu.org/licenses/lgpl.html>

- **PD:** This image is in the public domain.
- **ATTR:** The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- **EURO:** This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- **LFK:** Lizenz Freie Kunst. <http://artlibre.org/licence/lal/de>
- **CFR:** Copyright free use.
- **EPL:** Eclipse Public License. <http://www.eclipse.org/org/documents/epl-v10.php>

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses³³. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrowser.

27. Licenses

27.1. GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc.
<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this

way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that

they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of

products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those cases that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary.

To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily lia-

ble for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a

prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in a normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essen-

subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions

so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WI-

PO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep in-

tact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy.

This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work

in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Copying Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product

that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities,

provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer

product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the producer has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or vio-

lates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modi-

fied versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this Licen-

se, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to re-

ceive new licenses for the same material under section 10.9. Acceptance Not Required for Hanging Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing comp-

liance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any pa-

tent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide,

royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) ar-

range to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or

other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a

work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any

later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS

AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,

INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty, and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <<http://www.gnu.org/licenses/>>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <<http://www.gnu.org/philosophy/not-lgpl.html>>.

27.2. GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002,
2007, 2008 Free Software Founda-
tion, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0.
PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which

is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. **APPLICABILITY AND DEFINITIONS**

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License.

Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual

or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A Secondary Section is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philoso-

phical, ethical or political position regarding them.

The Invariant Sections are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising

the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats

include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in

parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section Entitled XYZ according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication on that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this Li-

ense applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify

you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transpa-

rent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Mo-

dified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- * A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediate-

ly after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "His-

tory section. You may omit a net-work location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled Acknowledgements or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled Endorsements.

Such a section may not be included in the Modified Version. * N. Do not retitling any existing section to be Entitled Endorsements or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no

material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement

made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are

multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled Endorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you

follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an aggregate if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of the Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions

of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your right holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you

of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from

time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License or any later version applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is eligible for relicensing if it is licensed under this License, and if all works that we-

re first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME; Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with

no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

27.3. GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc.
<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are

based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

* b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library

among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information

is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other libra-

ry facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Li-

brary as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser Gene-

ral Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's

public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.