Excel-VBA

Dieses Buch wurde aus den fogenden Büchern zusammengefasst:
- **Excel 2000 programmieren** von Addison-Wesley
- **Excel-VBA-Programmierung** von Markt+Technik
- **Office 2000 Developer Edition** von Markt+Technik
- **Jetzt lerne ich VBA mit Office 2000** von Markt+Technik
- **Jetzt lerne ich VBA mit Excel** von Markt+Technik
- **Excel 2000 Visual Bassic Schritt für Schritt** von Microsoft Press
- **Excel 2000 für Windows. Automatisierung / Programmierung**. vom Herdt Verlag
- **VBA-Programmierung.**
  Integrierte Lösungen mit Office 2000. Backoffice. vom Herdt Verlag

und sogar auch aus dem Buch:
- **VBA mit Word 2000 lernen** von Addison-Wesley

Überblick

1 Die Entwicklungsumgebung

1.1 Menüs im VBA-Editer

Menü bearbeiten

- Eigenschaften und Methoden anzeigen ***CTRL+J***
- Konstanten anzeigen ***CTRL+Shift+J***
- Quickinfo ***CTRL+I***
- Parameterinfo ***CTRL+Shift+I***
- Wort Verfolständigen ***CTRL+Leertaste***
- Bearbeiten /
  - Lesezeichen setzen / zurück setzen
  - Nächstes Lesezeichen
  - Vorheriges Lesezeichen
  - Alle Lesezeichen löschen
    - Setzt den Coursor in die nächste Programmzeile, die ein Lesezeichen enthält

---

Menü Ansicht

- Code F7
- Obekt Shift+F7

- Definition ***Shift+F2***
- Letzte Position ***CTRL+Shift+F2***

- Objektkatalog F2
  - Ansicht / Objektkatalog (F2)
  - Mit Extras / Verweise können Sie weitere Mappen bekannt geben, deren Prozeduren Sie nutzen wollen und die anschliessend ebenfalls hier im Objektkatalog aufgelistet werden.
  - Wählen Sie im Linken Listenfeld ein Modul aus und doppelklicken Sie im rechten Listenfeld auf eine Prozedur, die darauf hin angezeigt wird.
  - Zusätzlich befinden sich im Listenfeld u.a. die Einträge **VBA**, **Excel** und **MSForms** für die **VB-Bibliothek**, die **Excel-Bibliothek** und die **Formularobjekte-Bibliothek**.
  - Sie können die Suche nach einer Prozedur erleichtern, indem Sie im zweiten Listenfeld einen Teil des Prozedurnamens eintippen, z.B. vb.
  - Die VBA-Bibliothek ist leider unvollständig. Es fehlen alle Schlüsselwörter zur prozeduralen Programmierung (möglicherweise nicht in das Katalogkonzept integrierbar).
    Es fehlen auch ganz normale Fuktionen, die schlicht und einfach vergessen wurden (z.B. InStr)
  - Im Objektkatalog – Function Copy **([Description])**
    bsp. Sheets(1).Range(”A1”).Copy Sheets(2).Range(”A2”)

Direktfenster ***CTRL+G***

- ? für ”Print”
- ?Range(”A1:C2”).Cells(4).Adress
  $A$2

- Lokalfenster
- Überwachungsfenster
  - Eine ausgefeilte Möglichkeit zur Definition von Haltepunkten stellen Überwachungsausdrücke dar. Dabei handelt es sich zumeist um einfache Variablen oder Eigenschaf-

ten, deren Zustand überwacht werden soll. (Erlaubt sind aber auch einfache zusammengesetzte Ausdrücke.) Eingabe der Überwachungsausdrücke mit Kontexmenü ”Überwachung hinzufügen”.

- Sie können zwischen drei Formen der Überwachung auswählen: Die einfachste Variante lautet ”Ausdruck überwachen”, d.h., VB zeigt den aktuellen Wert bei einer Programmunterbrechung im Testfenster an. Bei den beiden anderen Varianten kommt es zu einer Programmunterbrechung, wenn der gesamte Ausdruck den Wahrheitswert *True* annimmt oder sich nur ändert. Sie können Überwachungsausdrücke also dazu verwenden, um ein Programm automatisch zu unterbrechen, sobald eine Variable grösser als 100 wird.
- Ist für die ständige Überwachung beliebiger Ausdrücke während der Programmausführung zuständig. - Ausdruck: 2 * (a + b) -
- Nach der Definition wird automatisch das ”Überwachungsfenster” eingeblendet, dass Sie auch mit Ansicht / Überwachungsfenster’ einblenden können.
- Im Überwachungsfenster in der Spalte ”Kontext” wird der Modulname und der Prozedurname angezeigt, um mehrdeutige Variablen eindeutig zu erkennen.
- Sie können in beiden Listenfeldern die Werte ”(Alle Prozeduren)” bzw. (”Alle Module”) auswählen. Dann ist es Excel egal, wo sich die Variable a befindet. Wird garade die Prozedur Demo des Moduls Test ausgeführt, wird der Wert dieser Variablen a angezeigt. Wird jedoch gerade die Prozedur ProgrammTest des Moduls VB_Einführung ausgeführt, wird eben der Wert jener gleichnamigen Variable a angezeigt.
- Überwachungsarten:
  - **Überwachungsausdruck**. Der aktuelle Wert des betreffenden Ausdrucks wird angezeigt.
  - **Unterbrechen, wenn der Wert True ist**. Unterbrecht die Programmausführung, wenn der Ausdruck den Wahrheitswert True annimmt. Geben Sie z.B b = 14 ein. Es wird unterbrochen, wenn b gleich 14 bzw. True ist.
  - **Unterbrechen, wenn der Wert geändert wurde**. …
- Um eine Überwachung zu ändern, klicken Sie im Überwachugsfenster auf die betreffende Zeile und wählen danach Debuggen / Bearbeiten. Das gleiche Dialogfeld wie bei Debuggen / Überwachung hinzufügen erscheint.

Aufrufeliste ***CTRL+L***

- Sie können ein Dialogfenster anzeigen, das alle Vorläuferprozeduren aufzählt, die zum Aufruf der gerade aktuellen Prozedur geführt haben.

Projektexplorer ***CTRL+R***
Eigenschaftsfenster ***F4***

- Werkzeugsammlung
- Aktivierreihenfolge

- Symbolleiste
  - Bearbeiten
  - Debuggen
  - UserForm
  - Voreinstellungen
  - Anpassen

- Microsoft Word

Menü Einfügen

- Prozedur…

- Userform
- Modul
- Klassenmodul
- Datei…

Menü Format

- ---- Alles Befehle für UserForm und Steuerelemente ----

Menü Debuggen

- Debuggen ”Entwanzen” ist ein historisch bedingter Ausdruck für die Entwicklung und Behebung von Programmfehlern, da angeblich in einem der ersten Rechner eine Wanze, die sich einschlich, für Fehlfunktionen sorgte.
- Kompilieren von Projekt

- Einzelschritt ***F8***
- Prozedurschritt ***Shift+F8***
  - Führ normalerweise wie F8 nur eine einzige Anweisung aus. Wenn in dieser Anweisung allerding ein Unterprogramm oder eine Funktion aufgerufen wird, wird diese Prozedur als Ganzes sofort ausgeführt.
- Prozedur abschliessen ***CTRL+Shift+F8 (Wie F5?)***
  - Führt alle Anweisungen bis zum Ende der aktuellen Prozedur aus. Wenn dabei andere Prozeduren aufgerufen werden, werden auch diese vollständig ausgeführt.
- Ausführen bis Coursor-Position ***CTRL+F8***
  - Erspart das setzen eines Haltepunkts. Führt alle Programmzeilen ab der aktuellen bis zu jener, in der sich momentan der Cursor befindet, im Normalmodus aus, ohne Unterbrechung. Erst bei jener Programmzeile wird wieder angehalten.

- Übewachung hinzufügen
- Überwachung bearbeiten ***CTRL+W***
- Aktuellen Wert anzeigen ***Shift+F9***
  - Zeigt den aktuellen Wert eines zuvor markierten Ausdrucks an.

- Haltepunkt ein/aus ***F9***
- Haltepunkt löschen ***CTRL+Shift+F9***

- Nächste Anweisung festlegen ***CTRL+F9***
  - Ignoriert alle Programmzeilen ab der aktuellen bis zu jener, in der sich mementan der Cursor befindet und die dadurch als nächste ausgeführt wird.
  - Hebt die nächste auszuführende Anweisung hervor und ist nützlich, wenn Sie sich momentan woanders befinden, sich also irgendeine andere Programmstelle anschauten.
- Nächste Anweisung anzeigen

Menü Ausführen

- Sub/UserForm ausführen ***F5***
- Unterbrechen ***CTRL+Unterbrechen (Esc?)***
- Zurücksetzen
  - Beendet das Programm vorzeitig. Wenn VBA blockiert ist.
- Entwurfsmodus beenden

Menü Extras

- Verweise
  - Um von Excel aus mit Word zu arbeiten. KK: MS Word
  - Bezug auf Prozedur in anderer Arbeitsmappe, sogar, wenn nicht geöffnet. Extras / Verweise enthält für jedes geöffnete Projekt, ausser das eigene, einen eigenen Eintrag "VBA-Projekt", Ist das Projekt im Augenblick nicht geöffnet, klicken Sie auf durchsuchen.
  - Mit den beiden Pfeilen geben Sie die Priorität einer Bibliothek an. Verwenden Sie Objekte einer bestimmten Bibliothek sehr häufig, können Sie die Suche beschleunigen, in dem Sie der betreffenden Bibliothek eine höhere Priorität geben.
- Zusätzliche Steuerelemente

- Makros

- Optionen…
  - Register: Editor / KK: Automatische Datentipps
    c = a + b (a und b markieren) – automatischer Daten-tip: [a + b = 21]
  - Register: Allgemein / Option: Bei Bedarf
  - Register: Allgemein / Option: Im Hintergrund
  - Beide Optionenen aktiviert (Defaulteinstellung) bedeutet, dass sofort mit der Programmausführung begonnen wird, und nur jene Prozeduren kompiliert werden, die gerade benötigt werden. Der Vorteil: ein rascher Programmstart. Der Nachteil: manche offensichtliche Fehler werden erst spät gemeldet.
    Bei grösseren Projekten ist es zumeist sinnovoll, die Option zu deaktivieren, weil dann eventuelle Syntaxfehler im Code sofort gemeldet werden (und nicht irgendwann später, wenn die Prozedur erstmalig benötigt wird).
- Digitale Signatur…
  Erstellen über Win-Start / Ausführen / Selfcert.exe / Zertifikatsname geben / fertig

---

Menü Add-Ins

- Add-In-Manager

Kein Kompilieren bei Bedarf

- Kompilieren ist der Vorgang der Übersetzung des Quellcodes in ausführbaren Code. Wenn Sie nach Bedarf kompilieren, bedeutet das Folgendes:
- Eine Kompilierung findet nur dann statt, wenn der Code auch tatsächlich ausgeführt wird (und zwar direkt vorher). Dabei wird jeweils eine gesamte Prozedur oder Funktion überprüft.
- Kompilieren bezeichnet aber nicht nur den Vorgang des Übersetzens, sondern führt auch eine Reihe von Überprüfungen des Codes durch, die wesentlich umfangreicher als der Syntaxcheck in der Entwicklungsumgebung sind. Findet der Compiler einen Fehler, dann werden Sie mit der zugehörigen Fehlermeldung genau in die Zeile gesetzt, in welcher der Fehler aufgetreten ist.
- Kompilieren Sie nicht bei Bedarf (sondern immer den gesamten Code bevor das Programm gestartet wird), dann startet das Programm erst dann, wenn der gesamte Code vom Compiler als fehlerfrei bewertet wird. Der Vorteil bei dieser Vorgehensweise liegt darin, dass Fehler, die sonst erst zur Laufzeit offensichtlich werden (wenn Sie nämlich in die fehlerhafte Routine springen), bereits vor Programmstart angezeigt werden. Dadurch sparen Sie viel Zeit beim Testen
- Die Option *Variablendeklaration erforderlich* kann ihre volle Wirkung erst dann entfalten, wenn auch die Kompilierung immer vor Programmstart stattfindet!

- Warum wird diese Option nicht vornherein sinnvoll gesetzt? Es gibt einen kleinen Performanceverlust, weil jetzt bei Programmstart jedes Mal der gesamte Code kompiliert werden muss. Da VBA-Projekte aber meistens nicht besonders umfangreich sind (im Vergleich zu ”richtigen” Anwendungen) und die heute üblichen Rechner schnell genug sind, ist dieser Zeitaufwand fast vernachlässigbar.

Überwachungsfenster

- Ansicht / Überwachungsfenster
- Markieren Sie eine Variable im Codefenster mit der Maus und ziehen Sie diese in das Überwachungsfenster hinein. Es wird dort ein entsprechender Eintrag erzeugt. Wiederholen Sie diesen Schritt für alle relevanten Variablen.
- Arbeiten Sie den Programmcode im Einzelschritt-Modus ab und überprüfen Sie, wie sich die Variableninhalte im Überwachungsfenster ändern

Direktfenster

```
?Range(Range(”A1”), Range(”C1”).Cells.Count
3

?ActiveSheet.Range(Cells(1,1), Cells(3,2)Count
6
```

Diverses

- Sie dürfen maximal 10 Zeilen Code durch ”_” voneinander trennen

```
'*************************************************************************
'*
'*                              Kommentar                                '*
'*
'*************************************************************************
```

1.2 Syntaxzusammenfassung

Tastenkombinationen im VBA-Editor

CTRL+N fügt eine neue Zeile obehalb des Coursors ein

F1	Ruft die Hilfe auf
F2	Zeigt den Objektkatalog an
F3	Öffnet in einem Programmcodefenster den *Suchen*-Dialog
F4	Öffnet das Eigenschaftenfenster, sofertn zuvor ein Objekt selektiert wurde
F5	Startet die VBA-Prozedur, in der sich die Textmarke zur Zeit befindet
F7	Schaltet bei einem Benutzerformular zum Programmcodefenster um
F8	Führt den nächsen Befehl einer VBA-Prozedur aus
F9	Setzt an der aktuelle Position der Textmarke im Programmcodefenster einen Haltepunkt oder hebt ihn wieder auf
CTRL+E	Exportiert das aktuelle Modul
CTRL+M	Importiert das aktuelle Modul

Shift+F2	Zeigt die Definition eines Objektbegriffs im Objektkatalog oder einer Prozedur im Modulfenster
Shift+F9	Zeigt während einer Programmunterbrechung den aktuellen Wert des Ausdrucks an, auf dem sich die Textmarke befindet

Wechsel des aktuellen Fensters:

Alt+F11	wechselt zwischen Excel und der Entwicklungsumgebung
CTRL+Tab	wechselt zwischen allen Visual-Basic-Fenstern
Alt+F6	wechselt zwischen den beiden zuletzt aktiven Fenstern
CTRL+G	ins Direktfenster (Debugfenster) wechseln
CTRL+R	ins Projektfenster wechseln

Eigenschaftsfenster:

Shift+Tab	springt ins Objektlistenfeld
CTRL+Shift+x	springt mit dem Anfangsbuchstaben X

Programmausführung:

F5	Programm starten
CTRL+Unt. (Esc)?	Programm unterbrechen
F8	ein einzelnes Komando ausführen
Shift+F8	Kommando / Prozeduraufruf ausführen
CTRL+F8	Prozedur bis zur Cursorposition ausführen
CTRL+Shift+F8	aktuelle Prozedur bis zum Ende ausführen
F9	Haltepunkte setzten
CTRL+F9	Ort des nächsten Kommandos bestimmen

Codefenster:

Tab	markierten Zeilenblock einrücken
Shift+Tab	markierten Zeilenblock ausrücken
CTRL+Y	Zeile löschen
Shift+F2	zur Prozedurdefinition bzw. zur Bariablendeklaration
CTRL+Shift+F2	zurück zur letzten Cursorposition
F6	Codeausschnitt wechseln (bei zweigeteiltem Fenster)
CTRL+F	Suchen
F3	Weitersuchen
CTRL+H	Suchen und Ersetzen
CTRL+Leertaste	Schlüsselwort / Variablennamen vervollständigen
Tab	Auswahl im IntelliSense-Listenfeld durchführen
Esc	InteliSense-Listenfeld verlassen

2 Datentypen, Variablen und Konstanten

Datentypen

Typenkenn-zeichen	Datentyp	Variablentyp	Bereich	Speicherplatz (in Byte)	Präfix
	Ja / Nein	Boolean	True / False	2	f
	Ganzzahlen	Byte (Ganzzahlen)	0-255	1	byt
%		Integer (Ganzzahlen)	± 32’767	2	int
&		Long (längere Ganzzahlen)	± 2’147’483'647	4	lng
!	Dezimalzahlen	Single (Gleitkommazahl mit sechstelliger Genauigkeit)	± 3,402823E38	4	sng
#		Double (Gleitkommazahl mit zehnstelliger Genauigkeit)	± 1,797693134862 32	8	dbl
@		Currency (Festkommazahl / skalierte Ganzzahl)	± 922337203685477,5808	8	cur
	Datumszahlen	Date	1. Jan 100 – 31. Dez 9999	8	dtm
$	Text	String	Zeichenkette ca. 2 Milliarden Zeichen	10+Länge der Zeichen	str
	Wechselnde Typen	Variant	Beliebige Daten	22+Länge der Zeichen	var
		Object	Verweis auf ein Object		

Mit Präfixen erleichtern Sie es anderen, mit Ihren Makros weiterzuarbeiten

Variant

- `x = "12"`
  `y = x-10`
  VBA nimmt eine automatische Typkonvertierung vor.
- Variant ist zwar sehr komfortabel, aber VBA ist ständig mit der Überprüfung von Datentypen und gegebenenfalls mit entsprechenden Typumwandlungen beschäftigt, was Rechenzeit kostet.

Deklarationszwang

- Ich empfehle Ihnen dringend, dafür zu sorgen, das VBA Sie dazu zwingt:
  `Option Explicit`
- Zugegeben: Wen man etwas ausprobieren möchte, und es schnell gehen muss, so verzichtet man auf eine Variablendeklaration. Allerdings sollten Sie Variablen in grösseren Projekten immer deklarieren. Es erleichtert die Fehlersuche und vermeidet Fehler.
- `Dim a, b, c As Integer`
  Nur c ist eine Integerzahl, a und b haben den Datentyp Variant!

String fester Länge

- `Dim Variablenname As String * Länge` `"Mario……."`
  `Dim s As String * 10`

- `Dim Variablenname As Datentyp, Variablenname As Datentyp …`
  `Dim x As Double, y As String, z As String * 10`
  - y ist eine Stringvariable variabler Länge
  - z eine Stringvariabel fester Länge von zehn Zeichen

Zwei Buchstaben bezeichnen die Objektbibliothek

- xl… – Excel-Konstanten
- vb... – VBA-Konstanten `(vbOKOnly, vbOKCancel, vbQueston)`

Geltungsbereich von Variabeln und Konstanten:

- Im Prozedurlistenfeld den Eintrag "(Deklaration)" wählen
  - `Dim Variablenname As Datentyp`
  - `Private Variablenname As Datentyp`

  Besser mit *Private* kennzeichnen, da `Dim` auch nur für aktuelles Modul gültig ist.

- `Public Variablenname As Datentyp`
  Keine Private Variable des betreffenden Moduls mehr, sondern eine öffentliche Variable, die allen Prozeduren zur Verfügung steht (Ausnahme: Klassenmodule)
- Die Variable behält ihren Wert, bis das letzte Makro beendet ist
- Die Deklaration einer Variable mit `Static` bezieht sich im Gegensatz zur Deklaration einer Prozedur mit `Static` nur auf die entsprechend deklarierte Variable und nicht auf alle Prozedurvariablen gleichzeitig.

Konstantendeklaration

- In Prozedur – `Const MWSt = 16`
- Im Deklarationsbereich – `Const MWSt = 16`
  oder `Public Const MWSt = 16`
- (Für alle Module ausser Klassenmodule)

- `Const Konstantenname = Ausdruck`
  `Const Konstantenname As Typ = Ausdruck`
  `Const a As String = ”Hallo”`
- `[Public/Global] [Private] Const Konstantenname = Wert As Datentyp`

Diverses

- Variable mit *Set* einen Objektverweis zuweisen (*Set* = Setzen)
- Zeichenkette `x$ = ”Hallo”`
  `x$ = ”Gerd” & ” ” & ”Meier”`
- Variablendeklaration: `Dim, Public, Private, Static`
- Lokale, globale und statische Variablen, Konstanten und Prozeduren
- `Rem` Kommentar
- `Let` Variablenname = Ausdruck
  Der Variable Variablenname wird der Wert Ausdruck zugewiesen

Schlüsselwörter:

`Sub, Function, End, For, To, Next, Dim, As …`
Objekt-, Methoden- und Eigenschaftsnamen gelten in der Regel nicht als Schlüsselwörter

3 Objekte

- Insgesamt sind in Excel rund 150 Objekte definiert.

OnlineHilfe

- Microsoft Outlook-Objekte
- Microsoft Word-Objekte
- Microsoft Excel-Objekte
- Microsoft PowerPoint-Objekte
- Microsoft Access-Objekte

Eigenschaften und Methoden

- Eigenschaften: Bestimmen das Verhalten und das Aussehen des Objekts
- Methoden: Ändern den Wert eines Objekts oder führen mit den Daten des Objekts eine Aktion aus
- Eigenschaften sind am ehesten mit Variablen vergleichbar
- Methoden sind eher mit Prozeduren vergleichbar

Objekt.Methode

```
Workbooks(Mappenname).Activate
Workbooks("Demo.xls").Activate
```

Objekt.Eigenschaft

```
Worksheets("Tabelle1").Name
ActiveSheet.Name
ActiveSheet.Next.Name
ActiveSheet.Previous.Name
```

- `Variable = Objekt.Eigenschaft`
  Liest den Wert einer Eigenschaft und weist ihn einer Variablen zu
- `Objekt.Eigenschaft = Variable`
  Weist genau umgekehrt einer Eigenschaft den angegebenen Wert zu

- `Objekt.Cells(Zeilenindex, Spaltenindex)`
  `ActiveSheet.Cells(1,2)`
- `Zelle B1`
  `ActiveSheet.Cells(1,2).Value = 10`

Bezüge mit Objektvariablen

- `Set Variable = Objektbezug`
- Variable mit Set einen Objektverweis zuweisen – `Set` = Setzen

Syntax zum Zugriff auf Eigenschaften

```
Variablenname = Objektverweis.Eigenschaftsname
Objektverweis.Eigenschaftsname = Ausdruck
```

Syntax der With-Struktur

```
With Objektverweis
  .Eigenschaftsname
  .Methodenname
Ent With
```

Syntax der Variablendeklaration

```
Dim Variablenname As Datentyp, Variablenname As Datentyp
Dim VariablennameTypkennzeichen, VariablennameTypkennzeichen
```

Syntax der Deklaration von Objektvariablen

```
Dim Objektvariablenname As Objekttyp
```

Syntax des Zuweisens eines Objektverweises

```
Set Objektvariablenname = Objektverweis
```

Syntax des Zugriffs auf einzelne Objekte einer Auflistung

```
Auflistung(Index).Eigenschaftenname
Auflistung(Index).Methodenname
```

- `Dim x As Objekt`
- `Dim x As Sheet`

Nothing

- `Nothing` hebt den in einer Objektvariablen gespeicherten Bezug wieder auf.
- `Set x = ActiveSheet`
- `Set x = Nothing`

Eigenschaftsprozeduren – Property

- `Property` = Eigenschaft
- Wie Sie sicher schon bemerkt haben, verbergen sich hinter Eigenschaften und Methoden letzlich ganz einfach Prozeduren, die auf ganz bestimmte Objekte angewendet werden. Derartige Prozeduren können Sie auch selbst erstellen. Und zwar drei Typen von `Property`-Prozeduren.
- einen Bezug auf ein Objekt übergeben
- einer Eigenschaft einen Wert zuweisen
- den Wert einer Eigenschaft übergeben

```
[Private / Public] [Static] Property ”Set / Let /Get” Prozedurname
[(Argumentenliste)]
  [Anweisungen]
  [Prozedurname = Ausdruck]
End Property
```

Beispiele

```
Sub HoleBlattname()
Dim x As Variant
  Set x = ActiveSheet
  MsgBox x.Name
  MsgBox x.Next.Name
End Sub
```

Methode Add

- `Sheets.Add Type:=xlChart`
- `ActiveSheet.Name = ”Mein Diagramm”`

- `Sheets.Add(Type:=xlChart).Name = ”Mein Diagramm”`

- Die Methode Add wird nicht mehr wie ein Unterprogramm ohne Rückgabewert eingesetzt, sondern wie eine Funktion. Aus diesem Grund ist der Parameter jetzt eingeklammert.

TypeName

- Ermittelt den Objektnamen

```
?Typename(ActiveSheet.Range("A1:C2").Cells)
Range
```

3.1Syntaxzusammenfassung

Eigenschaften zum Zugriff auf aktive Objekte

ActiveCell	aktive Zelle in einem Tabellenlatt
ActiveChart	aktives Diagramm in Tabellenblatt / Fenster / Arbeitsmappe / Excel
ActiveDialog	aktiver (zur Zeit sichtbarer) Dialog in Excel
ActiveMenuBar	aktive (zur Zeit sichtbare) Menüleiste in Excel
ActivePane	aktiver Ausschnitt eines Fensters
ActivePrinter	eingestellter Drucker eines Fenster
ActiveSheet	aktives Blatt in Fenster / Arbeitsmappe / Excel
ActiveWorkbook	aktive Arbeitsmappe in Excel
SeleectedSheets	ausgewählte Blätter eines Fenster
Selection	ausgewählte Objekte in Blatt / Fenster / Arbeitsmappe / Excel; die Eigenschaft kann je nach Auswahl auf die unterschiedlichsten Objekte verweisen; am häufigsten wird Selection zum Zugriff auf die ausgewählten Zellen eines Tabellenblatts verwendet
ThisWorkbook	Arbeitsmappe, deren Code gerade ausgeführt wird
Me	zum Modul gehöriges Objekt (z.B. Worksheet, UserForm)

Dim Variable As Objekttyp	Platzhalter für Objekte
Dim Variable As New Objekttyp	Dabei wird das entsprechende Objekt gleich erzeugt.
Set Variable = Objekt	Variable verweist auf das angegebene Objekt
Set Variable = Nothing	löscht den Verweis (nicht Objekt)
Name = TypeName(Variable)	ermittelt den Objektnamen

Worksheet-Ereignisse

Activate	Blattwechsel
BevoreDoubleClick	Doppelklick
BeforeRightClick	Klick mit der rechten Maustaste
Calculate	Inhalt des Blatts wurde neu berechnet
Deactivate	Blattwechsel
Selection Change	Veränderung der Markierung

Chart-Ereignisse

Activate	Blatt- (oder Diagramm-) Wechsel
BeforDoubleClick	Doppelklick
BeforRightClick	Klick mit rechter Maustaste
Calculate	Diagramm wurde neu berechnet
Deactivate	Diagrammwechsel
DragOver	Zellbereich wird über Diagramm bewegt (aber noch nicht losgelassen)
DragPlot	Zellbereich wurde losgelassen
MousDown	Maustaste wurde gedrückt
MouseMove	Maus wird bewegt
MousUp	Maustaste wurde losgelassen
Resize	Diagrammgrösse wurde verändert
Select	Diagramm wurde ausgewählt (markiert)
SeriesChange	Veränderung der ausgewählten Datenreihe

4 Prozedurale Programmierung

- Gleichnamige Prozeduren in unterschiedlichen Modulen sind erlaubt. Beim Aufruf den Namen des Moduls voranstellen.
  `- Call [Modul1].DemoA`

Makrorekorder

- Mit dem Makrorekorder kommt man schnell an den Code, für den man sonst lange suchen müsste.

Makro ausführen

```
Call Prozedurname
oder einfach:
Prozedurname
```

Der Datentyp des Rückgabewerts von Funktionen

- `Function Func(a, b) As Double`
- `Function Fund#(a, b)`

Die Parameterliste

- `Function Func(a As Double, b As Double) As Double`
- `Function Fund#(a#, b#)`
- Mit Kennungszeichen deutlich kürzere und übersichtlichere Funktionsdefinitionen
- Aus Gründen der Effizienz und Zuverlässigkeit sollten für alle Parameter einer Prozedur Datentypen angegeben werden.

Wert- und Rückgabeparameter:

- ```
  Sub array_Makro1()
  Dim a%, b%
    a = 4: b = 6
    array_Makro2 a, b
    Debug.Print a, b
  End Sub
  ```

- ```
  Sub array_Makro2(x%, y%)
    x = x * 2
    y = y / 2
  End Sub
  ```

- Parameter `x` und `y` heissen **Rückgabeparameter**, weil sich eine Veränderung auf den Ursprung der Daten auswirkt. (In höheren Programmiersprachen wird diese Art der Datenübergabe als Referenz- oder Zeigerübergabe bezeichnet, weil nicht die eigentlichen Daten übergeben werden, sondern ein Verweis auf die Daten an die Prozedur.) Eine Wertrückgabe ist natürlich nur dann möglich, wenn beim Aufruf der Prozedur tatsächlich eine Variable angegeben wird. Beim Makroaufruf `array_Makro2 1, 2` kann keine Rückgabe erfolgen (1 und 2 sind Konstanten), ebenso wenig bei zusammengesetzten Ausdrücken, etwa *array_Makro2 a+1, b/c*

- Wenn Sie generell vermeiden möchten, dass die Prozedur die übergebenen Variablen verändern kann, müssen Sie in der Parameterliste der Prozedurdefinition das Schlüsselwort **ByVal** angeben. Der jeweilige Parameter gilt dann als Wertparameter und

tritt innerhalb der Prozedur wie eine eigenständige Variable auf. Eine Veränderung des Parameters in der Prozedur hat keinen Einfluss auf Variablen ausserhalb der Prozedur.

- `Sub array_Makro2(ByVal x%, ByVal y%)`

- **`ByVal`** = `AlsWert (ByValue)`
- **`ByVal`**`-Parameter wie statische Variable`

- `Sub Makro(x())`
- Felder gelten immer als Rückgabeparameter, das Schlüsselwort `ByVal` ist nicht erlaubt

- Sobald ein Parameter als optional gekennzeichnet wird, müssen auch alle weiteren Parameter in dieser Form gekennzeichnet werden. Zuerst alle nichtoptionalen dann alle optionalen Parameter.
- Optionale Parameter können jeden Datentyp aufweisen. `IsMissing` funktioniert allerdings nur für Variant-Parameter.

Variable Parameteranzahl:

- Sie können beliebig viele Parameter übergeben. Sie müssen das Schlüsselfeld `ParamArray` verwenden und ein `Variant`-Feld angeben. `ParamArray` verträgt sich nicht mit Optional. Sie müssen sich entweder für optionale Parameter oder für eine variable Parameteranzahl entscheiden. Alle Parameter von ParamArray sind Wertparameter. Eine Veränderung der ursprünglichen Variable ist daher nicht möglich.
- `Function sum(ParamArray x())`

Benannte Parameter:

```
Makro para1, , , , , ,  para8
Makro Para8:=Konstante
```

- Globale Variablen, die in Modulen definiert sind, haben dieselbe Lebensdauer wie statische Variablen – ihr Wert bleibt also erhalten, bis die Datei geschlossen wird.

- Private empfiehlt sich für all jene Prozeduren, die nur für den internen Gebrauch innerhalb eines Moduls konzipiert sind. Auch bessere Übersichtlichkeit in den Listen zur Auswahl von Makros.

- Anweisung: `Option Private Modul'` am Beginn des Moduls schränkt den Gültigkeitsbereich öffentlicher Variablen und Prozduren auf die aktuelle Arbeitsmappe ein.

- `Dim Variable` definiert lokale Prozedur- oder Modulvariable
- `Private Variable` hat bei der Variablendeklaration die gleiche Wirkung wie Dim
- `Public Variable` definiert globale Variablen (nur auf Modulebene möglich)

- `Like`: mit dem Sie Zeichenketten (”M*4” für ”Mayr”, ”Meier” oder ”Mayier”) erkennen
- `Not`: Wahrheitswert umdrehen

Zugriff auf Variablen und Prozeduren aus anderen Arbeitsmappen

- Erstens müssen Variablen in der anderen Arbeitsmappe als `Public` deklariert sein, und Prozeduren dürfen nicht als `Private` definiert sein. Zweitens muss über Extras / Verweise ein Verweis auf jene Arbeitsmappe eingerichtet werden,

- `Private Static Sub Test()`
- `Private Static Funktion Test`
- `Property = Eigenschaftsprozedur`

4.1Syntaxzusammenfassung

Prozedurdefinition:

Sub Makro([Parameterliste])
Function Funk([parameterliste]) [As Datentyp]

Definition der Parameterliste

para1, para2, para3	3 Parameter im Defaultdatentyp Variant
para As Datentyp	Parameter im angegebenen Datentyp
para() [As Datentyp]	Feld
ByVal para [As Datentyp]	Wertparameter
Optional para [As Datentyp]	Optionaler Parameter
ParamArray para()	Liste mit variabler Anzahl von Parametern

Prozeduraufruf

Makro x1, x2, x3	Herkömmliche Ang. aller Parameter (Unterprogramme)
Ergebnis = funktion(x1, x2, x3)	Herkömmliche Ang. aller Parameter (Funktion)
Makro Para1:=x1, Para3:=x3	benannte Parameter (Unterprogramme)
Ergebnis = funktion(para3:=x1)	benannte Paremeter (Funktion)

Definition von Variablen auf Modulebene

Dim Variable	Modulvariable
Private Variable	Modulvariable (wie Dim)
Public Variable	öffentliche Variable (alle Module)
Option Private Module	öffentliche Variablen nur innerhalb der Arbeitsmappe verwendbar (auch bei Verweis)

Definition von Variablen auf Prozedurebene

Dim Variable	Lokale, nur in Prozedur verwendbar
Static Variable	Wie oben, behält aber Wert andauernd
Static Sub/Function Name()	alle Variablen der Prozedur sind statisch

Definition von Prozeduren

Sub/Function name()	öffentlich, für alle Arbeitsblätter
Private Sub/Function name()	nur im aktuellen Modul verwendbar
Option Private Modul	öffentliche Prozeduren nur innerhalb der Arbeitsmappe verwendbar (auch bei Verweis)

4.2Kontrollstrukturen

Vergleich der For Schleife mit der For Each-Schleife

- In vielen Fällen ist der Einsatz einer For Each-Schleife viel komfortabler als der Einsatz einer For-Schleife. For-Schleifen sind jedoch universeller einsetzbar, da jede For-Each-Schleife mit einer For-Schleife reproduzert werden kann.
- Die For Each-Schleife ist eine Sonderform der For..Next-Schleife und zum konfortablen ”Durchlaufen” von Arrays und den später erläuterten ”Klassen” gedacht.

Syntax der For Each-Schleife

```
For Each Objektvariable In Auflistung
  …
Next
```

Syntax der For Schleife

```
Auflistung(Objektname)
Auflistung(Index)
x = Auflistung.Count

For Index = 1 To Auflistung.Count
  Ausdruck = Auflistung(i)
Next i

For Zähler = Anfangswert To Endwert
  Anweisungen
Next [Zähler]
```

For Schleife

```
For i = 1 To 100 Step 5
  Debug.Print i
Next i
```

Schleife mit Sprungmarke realisieren

```
Dim i As Integer
  i = 0
  Schleife:
  Debug.Print ”Hallo”
  i = i + 1
  If i < 10 Then GoTo Schleife
End Sub
```

While and Wend

```
While Bedingung
  Anweisungen
Wend

Dim x As Integer
 x = 1
  While x < 11
    Debug.Print x
    x = x + 1
```

```
  Wend
```

Do Loop

- Kopfschleife

```
Do [{While / Until} Aussage]
 [Block]
Loop
```

- Fusschleife

```
Do
  [Block]
Loop  [{While / Until} Aussage]

Do = Durchlaufe
While = Solange
Until = bis Wahr
```

Beispiele von Schleifenköpfen

- ```
  Do Until i = ActiveSheet.UsedRange.Rows.Count
    …
    i = i + 1
  Loop
  ```
- ```
  Do Wihle ActvieCell.Value <> ””
  ```
- ```
  For i = 1 To ActiveWorkbook.Sheets.Count
  ```
- ```
  For i = 1 To ActiveSheet.UsedRange.Columns.Count
  ```
- ```
  For x = 1 To 322 Step 3
  ```

- ```
  Auch ein abwärtszählen ist möglich
  For x = 322 To 1 Step -3
  ```

- ```
  Und sogar ein schrittweise Weiterzählen mit Dezimalzahlen
  For x = 322 To 1 Step -2.5
  ```

Exit

```
Exit {For / Do / Function / Sub}
```

Select Case

```
Case 98
Case 5, x, 30
Case Is >= 5
Case x To y
Case ”Meier”
Case ”a”, Name$
Case Is < ”Otto”     - alphabetisch kleiner als Otto
Case ”c” To ”n”
```

Laufbedingung

- ```
  Schleifen mit einer Laufbedingung (While) werden so lange ausgeführt, wie die Bedingung erfüllt ist.
  ```

Abbruchbedingung

- Schleifen mit einer Abbruchbedingung (Until) werden so lange ausgeführt, wie die Bedingung nicht erfüllt ist.

5 Bestimmten Zellenwert suchen

Kurze Formulierung: Erste freie Zelle in Spalte anspringen

Variante 1

```
Range(”A65536”).End(xlUp).Offset(1, 0).Select
```

Variante 2 (besser)

```
Cells(Rows.Count, 1).End(xlUp).Offset(1).Value = Eingabewert
```

Kurze Formulierung: Leere Zelle in Spalte finden

```
Set b = Range(”A:A”).Find(””)
```

Kurze Formulierung: Spalten nach Wert durchsuchen und leere Zelle in Zeile finden

```
Range(”1:1”).Find(aktie).EntireColumn.Find(””).Value = kurs
```

Auf heutigem Spaltendatum stoppen

```
Range(”A1”).Select
Do Until Date = ActiveCell.Value
  ActiveCell.Offset(0, 1).Select
Loop
```

Tabellenblatt durchsuchen

```
Sub DatenSuchen()
str = InputBox(”Bitte geben Sie den Suchbegriff ein!”)
If str = ”” Then Exit Sub
ActiveSheet.UsedRange.Select
For Each Zelle In Selection
  If Zelle = str Then
    Zelle.Select
    Exit Sub
Next Zelle
MsgBox ”Suchbegriff nicht gefunden!”
End Sub
```

Ganze Arbeitsmappe druchsuchen

```
Sub DatenSuchenInGanzerArbeitsmappe()
Dim Zelle As Range
Dim Blatt As Worksheet
Dim str As String
  str = InputBox(”Bitte geben Sie den Suchbegriff ein!”)
  For Each Blatt in ActiveWorkbook.Sheets
    Blatt.Activate
    ActiveSheet.UsedRange.Select
    For Each Zelle In Selection
      If Zelle = str Then
      Zelle.Select
      Exit Sub
      End If
    Next Zelle
  Next Blatt
```

```
End Sub
```

Text in Arbeitsmappe suchen und ersetzen

```
Sub TextInArbeitsmappeSuchenUndErsetzen()
Dim s_Such As String
Dim s_Ersetz As String
Dim Blatt As Worksheet
Dim Treffer As Range
Dim Treffer1 As Variant
  s_Such = InputBox(”Geben Sie den Suchbegriff ein!”)
  s_Ersetz = InputBox(”Geben Sie den Ersetzbegriff ein!”)
  For Each Blatt In Worksheets
    Blatt.Activate
    Set Treffer = Blatt.Cells.Find(s_Such)
    If Not Treffer Is Nothing Then
      Treffer1 = Treffer.Address
      Do
        Treffer.Activate
        Treffer.Value = s_Ersetz
        Set Treffer = Cells.FindNext(After:=ActiveCell)
        On Error Resume Next
        If Treffer.Address = Treffer1 Then Exit Do
        Treffer.Value = s_Ersetz
      Loop
    End If
  Next Blatt
End Sub
```

Erste Spalte nach Wert durchsuchen, der in Zelle A1 eingegeben wurde

- Es kann der erste, die esten beiden oder die ersten drei Buchstaben eingegeben werden

```
Sub SuchenBestimmteZeile
Dim s As String
Dim i As Integer
s = Range(”A1”).Value
Range(”A3”).Select
For i = 1 To ActiveSheet.UsedRange.Rows.Count
 If Left(ActiveCell.Value, 1) = s
   Or Left(ActiveCell.Value, 2) = s _
   Or Left(ActiveCell.Value, 3) = s _
   Or s = ActiveCell.Value Then Exit For
 ActiveCell.Offset(1, 0).Select
Next i
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
  If Target.Address = ”$A$1” Then SuchenBestimmteZeile
End Sub
```

Artikel suchen

```
Dim SuchNr As String, Zähler As Integer
  SuchNr = InputBox(”Bitte geben Sie eine Artikelnummer ein:”, _
  ”Artikel suchen”)
  ThisWorkbook.Sheets(”Artikel”).Activate
  For Zähler = 1 To Range(”A1”).CurrentRegion.Rows.Count
    If Cells(Zähler, 1).Value = SuchNr Then
```

```
      MsgBox ”Artikel Nr. ” & SuchNr & vbCrLf & Cells(Zähler,
2).Value & ” - ” & _
         Cells(Zähler, 3).Value & ” DM”, vbInformation
      Exit Sub
    End If
  Next
MsgBox ”Der gewünschte Artikel wurde nicht gefunden.”, vbCritical
```

Text in Arbeitsmappe suchen und kennzeichnen

```
Sub TextInArbeitsmappeSuchenUndKennzeichnen()
Dim s As String
Dim i As Integer
Dim Erg1 As Variant
Dim Erg2 As Variant

s = InputBox(”Geben Sie den Suchbegriff ein!”, ”Textsuche”)
If s = ”” Then Exit Sub

For i = 1 To ActiveWorkbook.Sheets.Count
  Sheets(i).Activate
  Set Erg1 = Sheets(i).Cells.Find(s)
  If Not Erg1 Is Nothing Then
   Erg2 = Erg1.Address
    Do
      With Erg1
       .Activate
       .Interior.ColorIndex = 4
      End With
      Set Erg1 = Cells.FindNext(After:=ActiveCell)
      On Error Resume Next
      If Erg1.Address = Erg2 Then Exit Do
      Erg1.Interior.ColorIndex = 4
    Loop
  End If
Next i
```

Arbeitsmappen miteinander vergleichen

```
Dim Mappe1, Mappe2 As Workbook
Dim Blatt As Worksheet
Dim Zelle As Object
Dim s As String

Set Mappe1 = ThisWorkbook
Set Mappe2 = Workbooks(2)

For Each Blatt In ThisWorkbook.Worksheets
  Blatt.Activate
  s = Blatt.Name
  For Each Zelle In Blatt.UsedRange
    If Zelle.Value <>
     Mappe2.Worksheets(s).Range(Zelle.Address).Value Then
      Zelle.Activate
      Debug.Print ”Unterschied in Zelle ” & Zelle.Address & ” auf _
        Tabellenblatt ” & Sheets(s).Name
    End If
  Next Zelle
```

```
Next Blatt
```

Direktsuche bei grösseren Datenbeständen mit der Methode Find

```
Dim Bereich As Range
Dim s As String
  s = InputBox(”Geben Sie den Suchbegriff ein!”)
  If s = ”” Then Exit Sub
  Set Bereich = Columns(1).Find(s, LookAt:=xlPart)
  If Bereich Is Nothing Then
    MsgBox ”Name '” & s & ”' konnte nicht gefunden werden!”
  Else
    Bereich.Select
  End If
```

Die komplette Befehlssyntax der Methode Find

`Ausdruck.Find(What, After, LooIn, LookAt, SearchOrder, SearchDirection, MatchCase, MatchByte)`

- Wenn Sie die Suche wiederholen möchten, um die nächste passende Zelle zu finden, können Sie nochmals `Find` aufrufen und dabei die letzte Ergebniszuelle im Parameter `after` angeben. Bequemer ist es aber, die Methode `FindNext` oder `FindPrevious` zu verwenden, wo Sie als einziger Parameter `after` angeben möchten.
- Solange es im Suchbereich eine Zelle gibt, die dem Suchkriterium entspricht, wird diese gefunden – auch dann, wenn sich diese Zelle *oberhalb* des Parameters *after* befindet. Aus diesem Grund liefern die folgenden Zeilen eine Endlosschleife, sofern es im Suchbereich eine einzige Zelle gibt, die die Zeichenkette ”xyz” enthält!

```
Dim obj As Oject
Set obj = [a1].CurrentRegion.Find(”xyz”)
Do Until obj Is Nothing
  obj.Interior.Color = RGB(196, 196, 196)        ‘grauer Hintergrund
  Set obj = [a1].CurrentRegion.FindNext(obj)     ‘suche nächste Zelle
Set obj = [a1]CurrentRegion.FindNext(obj)
```

- Entspricht dem Menübefehl: Bearbeiten / Suchen

What

- Der erste Parameter enthält den Suchtext, alle weiteren Parameter sind optional

After

- Nachfolgend. Gibt die Zelle an, nach der die Suche beginnen soll

LookIn

- SuchenIn. Dieses Argument bestimmt, wo überall gesucht werden soll. Möglich ist hierbei die Suche in Formeln (xlFormulas), in Zellwerten(xlValues) oder gar in Kommentaren (xlComments)

LookAt

- Suchvergleich. Bestimmt die Übereinstimmung des Suchbegriffs mit dem Scuhergebnis. Bei ”xlPart” muss nicht der komplette Suchbegriff mit dem Ergebnis übereinstimmen. Bei ”xlWhole” müssen der Suchbegriff und das Suchergebnis identisch sein.

SearchOrder

- Gibt an, in welcher Reihenfolge zuerst gesucht werden soll. Dabei spielt die Anordnung Ihrer Daten eine grosse Rolle. Verwenden Sie die Konstante ”xlByColumns”, wenn Sie Ihre Tabelle spaltenweise durchsuchen möchten, also von oben nach unten. Nehmen Sie die Konstante ”xlByRows”, wenn Sie die Suche zeilenweise von links nach rechts durchführen möchten. Die schnellere Suche ist normalerweise die Suche in Spalten

SearchDirection

- Suchrichtung. Gibt die Suchrichtung an. Dabei haben sie die Möglichkeit zwischen den Konstanten ”xlNext” (xlNächster) bzw. ”xlPrevious” (xlVorheriger) zu wählen

MatchCase

- Gibt an, ob zwischen Gross- und Kleinschreibung unterschieden werden soll

Beispiele: Datum und Kurs einschreiben

Datum erfragen und eintragen

```
Sub DatumErfragenUndEintragen()
  heute = CDate(InputBox(”Datum?”, , CStr(Date)))
  ’Fügt aktuelles Datum ein und konvertiert String- in Datumsformat
  Set b = Range(”A:A”)
  Set b = b.Find(””)
  b.Value = heute
End Sub
```

heute = CDate(InputBox(”Datum?”, , CStr(Date)))

- Die Anweisung wandelt der CStr-Funktion das von der Funktion *Date* übergebene aktuelle Datum in eine Zeichenkette um. Diese Zeichenkette wird der Funktion *InputBox* als drittes Argument übergeben, was bewirkt, dass diese Zeichenkette und somit das aktuelle Datum im resultierenden Dialogfeld vorgegeben wird.
- Die nach dem Schliessen von *InputBox* übergebene Zeichenkette wird danach genau umgekehrt mit der CDate-Funktion in ein Datum umgewandelt und der Datumsvariablen *heute* zugewiesen.

Set b = Range(”A:A”)

- In der Zuweisung übergibt der Ausdruck Range(”A:A”) einen Bereichsbezug auf die komplette Spalte A als Range-Objekt, der anschliessend der Range-Variablen b zugewiesen wird. Dieser Bereichsbezug wird im Ausdruck

Set b = b.Find(””)

- Der Find-Methode übergeben, die den angegebenen Bereich durchsucht, im Beispiel also die Spalte A

b.Value = heute

- Die leere Zelle wird nicht selektiert oder aktiviert. Dafür übergibt Find jedoch selbst einen Bereichsbezug, und zwar einen Bezug auf die gefundene Zelle. Er wird b zugewiesen, so dass b nun auf Zelle A7 verweist

Kurs erfragen und eintragen

```
Sub KursErfragenUndEintragen()
  aktie = InputBox(”Aktie?”)
  kurs = CSng(InputBox(”Kurs?”)
  Set b = Range(”1:1”)
  Set b = b.Find(aktie)
  Set b = b.EntireColumn
  Set b = b.Find(””)
  b.Value = kurs
End Sub
```

aktie = InputBox(”Aktie?”)

- Die von InputBox übergebene Zeichenkette, z.B ”523,3”, wird mit der CSng-Funktion in einen Single-Wert gewandelt und der Variablen *kurs* zugewiesen, die entsprechend vom Typ Single sein muss

Set b = Range(”1:1”)
- Weist b einen Bezug auf die komplette Zeile 1 zu. Anschliessend wird auf diesen Bezug die Find-Methode losgelassen, die Zeile 1 nach der in *aktie* enthaltenen Zeichenkette durchsuchen soll, z.B nach ”Daimler”:

Set b = b.Find(aktie)
- Ohne weitere Argumente unterscheidet *Find* nicht zwischen Gross-/Kleinschreibung und findet daher in Zelle C1 den Text ”DAIMLER”
- Der übergebene Bereichsbezug auf diese Zelle wird b zugewiesen. Nun muss die Spalte, in der sich dieser Aktienname befindet, nach der ersten leeren Zelle durchsucht werden.

Set b = b.EntireColumn
- Benutzt die EntireColumn-Methode, die auf Bereichsobjekte angewandt wird und selbst wieder einen Bereichsbezug übergibt, und zwar einen Bezug auf die komplette Spalte, auf die der Bezug verweist.

Set b = b.Find(””)
- Erneut wird *Find* angewendet. Der Bereichsbezug b, der nun auf Spalte C verweist, wird nach der ersten leeren Zelle durchsucht:
- Im Beispiel findet *Find* die Zelle C7 und übergibt einen Bereichsbezug auf diese Zelle, der wieder b zugewiesen wird. Die folgende Anweisung

b.Value = kurs
- Speichert den Inhalt von kurs in dieser Zelle.

Knappere Formulierung

```
Set b = Range(”A:A”)
Set b = b.Find(””)
```

Kürzer:
```
Set b = Range(”A:A”).Find(””)
```

Anstatt so:
```
heute = CDate(InputBox(”Datum?”, , CStr(Date)))
Set b = Range(”A:A”)
Set b = b.Find(””)
b.Value = heute
```

so:
```
heute = CDate(InputBox(”Datum?”, , CStr(Date)))
Set b = Range(”A:A”).Find(””)
b.Value = heute
```

Anstatt so:
```
  aktie = InputBox(”Aktie?”)
  kurs = CSng(InputBox(”Kurs?”)
  Set b = Range(”1:1”)
  Set b = b.Find(aktie)
  Set b = b.EntireColumn
```

```
Set b = b.Find(”“)
b.Value = kurs
```

so:

```
aktie = InputBox(”Aktie?”)
kurs = CSng(InputBox(”Kurs?”)
Range(”1:1”).Find(aktie).EntireColumn.Find(”“).Value = kurs
```

- Range übergibt einen Bezug auf einen Bereich, genauer auf Zeile 1. Auf diesen Bezug wird die `Find`-Methode angewandt, um in dem Bereich den Aktiennamen zu finden. Die `Find`-Methode übergibt selbst wieder einen Bereichsbezug, und zwar auf die gefundene Zelle.
- Auf diesen Bezug wird nun die Methode `EntireColumn` angewandt, die die gesamte zugehörige Spalte als Bereichsbezug überprüft, und auf diesen wird erneut `Find` angewendet, diesmal, um die erste leere Zelle der Spalte zu finden. `Find` übergibt den entsprechenden Bereichsbezug, und mit der Eigenschaft `Value` wird der Zelle der in `kurs` enthaltene Wert zugewiesen.

6 Zellen und Zellbereiche

6.1 Bereichsnamen

Bereichsnamen

- Wird ein Name in Excel erstmalig definiert, gilt die Definition für die gesamte Arbeitsmappe. Wird derselbe Name ein weiteres Mal in einem anderen Tabellenblatt definiert, gilt diese Definition lokal für dieses Tabellenblatt; die alte Definition gilt weiter für alle anderen Tabellenblätter. Die Folge: Es ist bisweilen nur extrem schwer nachzuvollziehen, ob eine namensdefinition via ActiveWorkbook.Names(…) oder via `ActiveSheet.Names(…)` angespreochen werden muss. Entscheiden Sie sich im Zweifelsfall für die erste Variante: Dor enthält die Names-Auflistung alle lokalen Definitionen (des gerade aktuellen Tabellenblatts) sowie alle globalen Definitionen, die nicht durch das aktuelle Tabellenblatt überlagert sind.

Bereich benennen

```
Dim Bereich As Range
Worksheets(”Tabelle1”).Activate
Set Bereich = Sheets(”Tabelle1”).Range(”A1:A10”)
   ActiveWorkbook.Names.Add _
     Name:=”BenannterBereich”, _
     RefersTo:=Bereich,                                    _
     Visible:=True
   Bereich.Select
```

- Die Methode `Add` weist folgende Argumente auf:
  - `Name`
  - `RefersTo` gibt an, auf welchen Zellbereich sich der Name beziehen soll.
  - `Visible` bestimmt, ob der Name für den Anwender sichtbar sein soll oder nicht

Name in Arbeitsmappe ausgeben

```
Dim nam As Name
On Error Resume Next
For Each nam In ActiveWorkbook.Names
  MsgBox ”Name: ” & nam.Name & Chr(13) _
  & ”Adresse: ” & nam.RefersToLocal
Next
```

Zellenwerte automatisch als Namen übernemen

- `ActiveWorkbook.Names.Add Name:=ActiveCell.Value`

Konstanten als Namen vergeben

- Sie können auch Namen vergeben, die sich auf keinen Zellbezug beziehen, sondern einen konstanten Wert beinhalten.
  `ActiveWorkbook.Names.Add Name:=”MwSt”, RefersTo:=”=1.16”`
- Auf den Namen MwSt können Sie jetzt überall in Ihrer Arbeitsmappe zugreifen.
  `=A3*MwSt`

Bezüge von benannten Bereichen ermitteln

```
Set Bereich = Range(Range(”BenannterBereich”).Address)
Debug.Print Bereich.Address(external:=True)
```

Namenprüfung einer Zelle

```
MsgBox ActiveCell.Name.Name
```

Benannten Bereich markieren

```
Dim s As String
    s = InputBox(”Bitte geben Sie den Namen ein!”, _
        ”Namensuche”)
    If s = ”” Then Exit Sub
    On Error GoTo Fehlermeldung
    Application.Goto Reference:=s
    Exit Sub
Fehlermeldung:
    MsgBox ”Der Name ” & s & _
    ” konnte in der Mappe nicht gefunden werden!”
```

Namen von Bereich Ermitteln

```
Dim Bereich As Range
  Set Bereich = Application.InputBox _
     (”Wählen Sie einen Zellenbereich aus!”, Type:=8)
   Range(Bereich.Address).Select
   On Error GoTo Fehlermeldung:
    MsgBox Selection.Name.Name
    Exit Sub
Fehlermeldung:
    MsgBox ”Für diesen Bereich ist kein Namen vergeben!”
```

Alle Namen protokollieren

```
Dim benannteBereiche As Object
Range(”A1”).Select
  For Each benannteBereiche In ActiveWorkbook.Names
    ActiveCell.Value = benannteBereiche.Name
    ActiveCell.Offset(0, 1).Value = _
       ActiveWorkbook.Names.Item(benannteBereiche.Name)
    ActiveCell.Offset(1, 0).Select
  Next
```

Namen ändern

- Alle verwendeten deutschen Zellnamen sollen in englische Zellennamen umgesetzt werden. U.a. muss eine Kundenliste umgesetzt werden, bei der der Zellenname aus dem eigentlichen Kundennamen mit der Vorsilbe ”Kunde” besteht. Diese Vorsilbe muss nun in ”Customer” umbenannt werden. Dabei müssen alle eingesetzten Namen in der Arbeitsmappe daraufhin überprüft werden.

```
Sub NamenÄndern()
Dim benannteBereiche As Object
Dim VglName As String

  For Each benannteBereiche In ThisWorkbook.Names
    VglName = benannteBereiche.Name
    MsgBox InStr(VglName, ”Kunde”)
```

```
    If InStr(VglName, ”Kunde”) > 0 Then
      VglName = Application.Substitute(VglName, ”Kunde”, ”Customer”)
      benannteBereiche.Name = VglName
    End If
  Next benannteBereiche
End Sub
```

- `InStr(Start, Durchsuchte Zeichenfolge, Gesuchte Zeichenfolge, _ [Vergleich])`

- Da Sie die Funktion `InStr` anwenden müssen, um die Namenprüfung durchzuführen, weisen Sie die Objektvariable ”BenannteBereiche” der Strin-Variablen `VglName` zu. Jetzt ermitteln Sie mit der Funktion `InStr`, ob der Textteil ”Kunde” im jeweiligen Namen vorkommt. Wenn ja, dann meldet die Funktion das erste Vorkommendes Textteils, einen Wert grösser Null, zurück. In diesem Fall kommt dann die Tabellenfunktion `Substitute` Einsatz. Dieser Tabellenfunktion übergeben Sie zuerst den Textteil, der ersetzt werden soll (Kunde), und danach die neue Zeichenfolge (Customer). Zum Schluss weisen Sie die überarbeitete String-Variable `VglName` der Objektvariable ”BenannteBereich” wieder zu und machen damit die Namensänderung perfekt.

Bezugsadresse von einem Namen ändern

```
Dim VerName As Name
 For Each VerName In ActiveWorkbook.Names
   If VerName.Name = ”BenannterBereich” Then
     VerName.RefersTo = ”=$A$1:$B$10”
     Range(”BenannterBereich”).Select
     Exit Sub
  End If
Next VerName
```

- Denken Sie daran, beim neuen Zellenbezug die Bezüge absolute zusetzen. Das Weglassen das Absolut-Zeichens `$` liefert keine zuverlässige Ergebnisse

In welchem Bereich steckt der Mauszeiger

```
Dim benannterBereich As Object
  For Each benannterBereich In ThisWorkbook.Names
    If Not Intersect(Selection, benannterBereich.RefersToRange) _
      Is Nothing _
    Then
      MsgBox Selection.Address & _
        ” ist innerhalb des benannten Bereichs ” _
        & benannterBereich.Name & Chr(13) & _
        ” und hat die Zellenadresse ” _
        & benannterBereich.RefersToRange.Address
      Exit For
    End If
 Next
```

- MsgBox $D$7 ist innerhalb des benannten Bereichs Bereich2 und hat die Zellenadresse $D$4:$D$10

- Mit der Methode `Intersect` wird geprüft, ob die momentan aktive Zelle innerhalb eines benannten Bereichs liegt. Diese Methode liefert den Wert `Nothing` zurück, wenn die aktuelle Zelle nicht in einem benannten Bereich liegt.

Namen verstecken

```
Dim benannteBereiche As Object
   For Each benannteBereiche In ActiveWorkbook.Names
       benannteBereiche.Visible = False
   Next benannteBereiche
```

Namen wieder sichtbar machen

```
Dim benannteBereiche As Object
   For Each benannteBereiche In ActiveWorkbook.Names
      benannteBereiche.Visible = True
   Next benannteBereiche
```

Namen löschen

- Einen:

```
ActiveWorkbook.Names(”BenannterBereich”).Delete
```

- Alle:

```
Dim benannteBereiche As Object
  For Each benannteBereiche In ActiveWorkbook.Names
     benannteBereiche.Delete
  Next
```

Alle Berechsnamen einer Mappe löschen

```
Dim nam As Name
  For Each nam In Application.Names
    nam.Delete
  Next nam
```

Varianten der Addressierung

Set R = Range(”E3”)

?R.Address(ReferenceStyle :=xlR1C1)
R3C5

?R.Address(ReferenceStyle :=xlA1)
$E$3

?R.Address(ReferenceStyle :=xlR1C1, RowAbsolute:=False, ColumnAbsolute:=True)
R[2]C[4]

?R.Address(ReferenceStyle :=xlR1C1, RowAbsolute:=True, ColumnAbsolute:=True)
R3C5

6.2 Kommentare

Kommentare erfassen

- Neben dem Namen noch zusätzlich das Datum und die Uhrzeit erfassen. Das folgende Makro fügt einen solchen Kommentar in der momentan aktiven Zelle ein.

```
Dim Kom As Comment
Dim s As String
 s = InputBox(”Geben Sie Ihren Kommentar ein!”, ”Kommentar erfassen”)
 If s = ”” Then Exit Sub
   Set Kom = ActiveCell.addcomment
   Kom.Text Application.UserName & Chr(10) & Date & Chr(10) & Time & ” Uhr” & _
     Chr(10) & s
  With Kom.Shape.TextFrame
  .Characters.Font.Name = ”Courier”
  .Characters.Font.Size = 12
  .AutoSize = True
  End With
```

Kommentare ergänzen

```
Dim sAlt As String
Dim sNeu As String
sNeu = InputBox(”Geben Sie einen Kommentar ein”, ”Kommentar ergänzen”)
If sNeu = ”” Then Exit Sub
With Selection
    On Error Resume Next
    sAlt = .Comment.Text
    If sAlt = ”” Then .addcomment
    sNeu = sAlt & Chr(10) & Application.UserName & Chr(10) & ”Kommentar vom ” _
      & Date & Chr(10) & sNeu & Chr(10)
    .Comment.Text sNeu
    .Comment.Visible = True
    .Comment.Shape.TextFrame.AutoSize = True
End With
```

Kommentare finden

- Das Makro markiert alle Zellen, die Kommentare enthalten. Zusätzlich sorgt es dafür, das der rote Indikator wieder angezeigt wird.

```
On Error Resume Next
 Selection.SpecialCells(xlCellTypeComments).Select
 If Application.DisplayCommentIndicator = 0 Then _
 Application.DisplayCommentIndicator = 1
```

- `xlNoIndicator 1`
- `xlCommentIndicatorOnly 2`
- `xlCommentAndIndicator 3`

Kommentare aus Zellentexten bilden

```
Sub KommentareAusZellenInhaltBilden()
Dim Kom As Comment
Dim Zelle As Object
  On Error Resume Next
  For Each Zelle In Selection
    Set Kom = Zelle.addcomment
    Kom.Text Date & Chr(10) & Zelle.Value
 Next Zelle
End Sub
```

Kommentare löschen

```
On Error Resume Next
  Selection.SpecialCells(xlCellTypeComments).Select
  Selection.ClearComments
```

Alle Kommentare einer Arbeitsmappe löschen

```
Dim i As Integer
Dim Notiz As Comment
  For i = 1 To Sheets.Count
    Sheets(i).Activate
    For Each Notiz In Sheets(i).Comments
      Notiz.Delete
    Next Notiz
 Next i
```

6.3 Diverses

Die Eigenschaft Range erforschen

```
Sub BereichBeobachten()
  Range(”A1”, ”D2”).Select
  Range(”ActiveCell, ”B6”).Select
  Range(”B2:C8”).Select
  Range(”B2:E4”).Name = ”Testbereich”
  Range(”Testbereich”).Select
  Range(”B2”).Select
  ActiveCell.Range(”B2”).Select            - ActiveCell bedeutet A1
  Range(”Testbereich”.Range(”A1”).Select   - Erste Zelle im
                                             Testbereich markieren
End Sub
```

Einen Bereich als Auflistung erforschen

```
Sub AuflistungBeobachten()
  Dim neuBereich As Range
  Set neuBereich = Range(”B2:E4”)
  neuBereich.Interior.Color = vbYellow
  neuBereich.Cells(1,4).Select
  neuBereich.Cells(6).Select
  neuBereich.Cells(neuBereich.Cells.Count).Select
  Cells(Cells.Count).Select
  neuBereich.Rows(2).Select
  neuBereich.Columns(neuBereich.Columns.Count).Select
  Columns(2).Select
End Sub
```

Berechnete Bereiche erforschen

```
Sub BerechnungBeobachten()
  Dim neuBereich As Range
  Sheets(”Preise”).Activate
  Set neuBereich = Range(”C4:E5”)
  neuBereich.Interior.Color = vbYellow
  neuBereich.Offset(1, 0).Select
  neuBereich.Offset(0, neuBereich.Columns.Count).Select
  neuBereich.Offset(-1, -1).Resize(neuBereich.Rows.Count _
    + 2, neuBereich.Columns.Count + 2).Select
  neuBereich.Cells(1).EntireRow.Select
  neuBereich.EntireColumn.Select
  neuBereich.CurrentRegion.Select
End Sub
```

Nach rechts verschieben

```
If ActiveCell.Column > 5 Then
  ActiveCell.Offset(0, 1).Select
Else
  Cells(ActiveCell.Row + 1, 1).Select
End If
```

Zellen vergleichen

```
Dim i As Integer
Calculate
For i = 1 To Range(”Überarbeiten”).Cells.Count
  If  Range(”Überarbeiten”).Cells.(i)  >  Range(”Original”).Cells.(i)
Then
    Range(”Überarbeiten”).Cells(i).Interior.Color = vbYellow
  Else
    Range(”Überarbeiten”).Cells(i).Interior.Color = vbCyan
  End if
Next i
```

Andere Schreibweise für Range

- `Sheets(”Tabelle1”).[a1:a4].Select`

Kopieren

- `Range(”A1”).Copy Range(”D1:D6”)`
- `Application.CutCopyMode = False`

Bereich begrenzen

Worksheets(”Tabelle1”).ScrollArea ”B2:G19”

Cells

- `Objekt.Cells(Zeilenindex, Spaltenindex)`
  `ActiveSheet.Cells(1,2)`
- `Zelle B1`
  `ActiveSheet.Cells(1,2).Value = 10`
- Wenn auf Zellen einer anderen Tabelle zugegriffen werden soll, lautet die korrekte Schreibweise nicht
  `Worksheets(n).Range(Cells(…), Cells(…))`

sondern
`Range(Worksheets(n).Cells(…), Worksheets(n).Cells(…))`

Reihenfolge (Zeile, Spalte) für Offset und Cells

- Sowohl `Offset` als auch `Cells` erwarten die Parameter in der Reihenfolge (`Zeile, Spalte`). Das wiederspricht sowohl der üblichen Nomenklatur von Zellen (etwa `B5`, wo zuerst die Spalte `B` und dann die Zeile `5` angegeben wird) als auch den mathematischen Gepflogenheiten (wo in der Regel zuerst die `X`-, dann die `Y`-Koordination angegeben wird).

NumberFormat, NumberFormatLocal und Style

- `NumberFormat` gibt als Zeichenkette das Zahlenformat an. `NumberFormatLocal` erfüllt dieselbe Aufgabve, allerdings wird die Zeichenkette in der landestypischen Schreibweise formatiert. `Style` verweist schliesslich auf eine Formatvorlage (`Style`-Objekt).
- Einige Formatvorlagen sind vordefiniert `(Builtin = True)`.

```
Dim s As Style
For Each s In ThisWorkbook.Styles
  If s.BuiltIn = True Then
    Debug.Print       s.Name,       s.NameLocal,       s.NumberFormat,
s.NumberFormatLocal
  End If
Next
```

Select und Activate

- `Select` wählt das angegebene Objekt aus
- `Activate` aktiviert das angegebene Objekt

Areas

- Die Methode ist ähnlich wie `Cells`, sie liefert aber zusammengehörige (rechteckige) Zellbereiche als Ergebnis. Die Anwendung von `Areas` ist zur Verarbeitung von Bereichen notwendig, die aus mehreren rechteckigen Teilbereichen zusammengesetzt sind (etwa nach einer Mehrfachauswahl mit CTRL).
- Bei zusammengesetzten Bereichen kann über Cells nur der erste rechteckige Teilbereich bearbeitet werden. Damit alle rechteckigen Bereiche bearbeitet werden können, muss die Methode **`Areas`** eingesetzt werden.
- Die Anwendung von Areas ist zur Verarbeitung von Bereichen notwendig, die aus mehreren rechteckigen Teilbereichen zusammengesetzt sind.

Union und Intersect

- Mit Range(Cells(…), Cells(…) können Sie nur einfache Bereiche definieren. Bereiche mit komplexer Form müssen mit **`Union`** aus mehreren rechteckigen Bereichen zusammengesetzt werden.
- **`Union`** und **`Intersect`**: Die beiden Methoden bilden aus mehreren Bereichen einen zusammengesetzten Bereich (**Vereinigung**) bzw. ermitteln jenen Bereich, der in allen angegebenen Bereichen vorkommt (**Schnittmenge**). Für erfahrene Programmierer: Union entspricht dem logischen Oder, Intersect dem logischem Und. Intersect eignet sich beispielsweise dazu, aus einem Bereich alle Zellen auszuwählen, die in einer bestimmten Zeile oder Spalte liegen. Mit Union können Sie aus mehreren rechteckigen Bereichen einen zusammengesetzten Bereich bilden.

Union - Die übersichtliche Mehrfachauswahl

- Wenn Sie mehr als zwei Bereiche markieren möchten, wird die Schreibweise `Range("A1:C9,E6:H14,A17:D23,G18:K27").Select` etwas unübersichtlich. Für eine übersichtliche Schreibweise setzen Sie die Methode `Union` ein.

```
Sub MehrereBereicheMarkieren()
Dim Ber1 As Range, Ber2 As Range
Dim Ber3 As Range, Ber4 As Range
Dim Bereiche As Range
    Worksheets("Tabelle1").Activate
    Set Ber1 = Range("A1:A5")
    Set Ber2 = Range("C1:C5")
    Set Ber3 = Range("A10:A15")
    Set Ber4 = Range("C10:C15")
    Set Bereiche = Union(Ber1, Ber2, Ber3, Ber4)
    Bereiche.Select
End Sub
```

AddressLocal und ConvertFormula

- **`AdressLocal`** funktioniert wie **`Adress`**, liefert Adressen aber in der Schreibweise der jeweiligen Landessprache (d.h. Z1S1 statt R1C1).
- Wenn Sie eine Adresse einmal besitzen, können Sie sie mit der Application-Methode **`ConvertFormula`** weiterverarbeiten. `ConvertFormula` ermöglicht unter anderem eine Konversion zwischen A1- und R1C1-Notation, zwischen absoluter und relativer Adressierung etc.

Notizen / Kommentare

- Der Zugriff auf eine Zellennotiz erfolgt über die Methode `NoteText`. Da mit Excel-Metoden nur Zeichenketten mit maximal 255 Zeichen übergeben werden können, weist diese Methode zwei Parameter auf, mit denen die Start- und Endposition innerhalb der Notiz angegeben werden kann. Diese Parameter ermöglichen es, auch Notizen, die länger als 255 Zeichen sind, auszulesen bzw. zu verändern.
- Seit Excel 97 heissen Notizen auch Kommentare. Sie werden jetzt über das `Comment-Objekt` verwaltet. (NoteText kann aber weiter verwendet werden). Mit der Methode `AddComment` können neue Kommentare definiert werden. `ClearComment` löscht vorhandene Kommentare. Die Aufzählung `Comments` für das `WorkSheet`-Objekt hilft beim Aufspüren aller Kommentare in einem Tabellenblatt.

Zelle auf Werte prüfen

```
If IsNummeric(ActiveCell.Value)
```

Zelle auf Datumswerte prüfen

```
If IsDate(ActiveCell.Value)
```

Prüfen, ob Zelle einen Wert enthält

```
If ActiveCell.Value = ""
eleganter:
If IsEmpty(ActiveCell)
```

Schrift und Hintergrundfarbe

```
If ActiveCell.Font.Bold = True
If ActiveCell.Font.ColorIndex = 3
If ActiveCell.Font.Italic = True
ActiveCell.Interior.ColorIndex = xlColorIndexNone     - Kein
```

Gross- und Kleinschreibung

- Die Funktion **`UCase`** (gross) unterscheidet nicht zwischen Gross- und Kleinschreibung. Sie wandelt Kleinbuchstabe automatisch in Grossbuchstaben um.
- Analog dazu wandelt die Funktion **`LCase`** (klein) Grossbuchstaben in Kleinbuchstaben um.

```
Select Case UCase(ActiveCell.Value)
  Case Is = ”A”, ”B”, ”C”
    …
```

Kopien entfernen

```
If Right(ActiveCell.Value, 5) = ”Kopie” Then …
```

Sortieren mit der Methode Sort

```
Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, _
  OrderCustom, MatchCase, Orientation, SortMethod)
Selection.Sort  Key1:=Range(”A1”), _
           Order1:=Ascending, _
           Header:=xlGuess, _
           OrderCustom:=1 _
           MatchCase:=False _
           Orientation:=xlTopToBottom
```

- Die ersten beiden Argumente müssen Sie immer auf einen Blick betrachten. Das erste Argument **`Key`** bestimmt das Sortierfeld, das zweite Argument **`Order`** die Sortierreihenfolge. Entweder aufsteigend (xlAscending) oder absteigend (xlDescending). Insgesamt können Sie drei verschiedene Sortierfelder bestimmen.
- Das Argument **`Type`** ist nur für Pivot-Tabellenbereiche interessant
- Das Argument **`Header`** legt fest, ob die erste Zeile Überschriften enthält oder nicht. Mit **`xlGuess`** überlassen Sie die Entscheidung Excel selbst. Setzten Sie die Konstante **`xlYes`**, wenn der Sortierbereich eine Überschriftenzeile enthält, die natürlich nicht mitsortiert werden darf. Wenn Sie die Konstante **`xlNo`** zuweisen, enthält der Sortierbereich keine Überschriften.
- Das Argumet **`CustomOrder`** wird bei benutzerdefinierten Sortierreihenfolgen verwendet
- Das Argument **`MatchCase`** (GrossKlein) nimmt den Wert `True` an, wenn beim Sortieren Gross- und Kleinschreibung berücksichtig werden soll. Mit `False` wird die Gross- und Kleinschreibung nicht berücksichtigt.
- Beim Argument **`Orientation`** wird die Sortierreihenfolge festgelegt. Hat das Argument den Wert **`xlSortRows`**, so wird von oben nach unten, also zeilenweise sortiert. Wird das Argument auf **`xlSortColumns`** gesetzt, so wird von links nach rechts, also spaltenweise sortiert.

Ausschneie- bzw. Kopiermodus entfernen

```
Application.CutCopyMode = False
```

Relative Markierungsform

```
Range(ActiveCell(), ActiveCell.Offset(5, 10)).Select
```

Bestimmte Zellen ansteuern mit der Methode *SpecialCells*

xlCellTypeAllFormatConditions
- Formatierte Zellen

xlCellTypeAllValidation
- Mit Gültigkeitsregeln

xlCellTypeBlanks
- Leere Zellen

xlCellTypeComments
- Kommentare

xlCellTypeConstants
- Konstanten

xlCellTypeFormulas
- Formeln

xlCellTypeLastCell
- letzte Zelle

xlCellTypeSameFormatConditions
- Zellen mit gleichem Format

xlCellTypeSameValidation
- Zellen mit gleichen Gültigkeitskritieren

xlCellTypeVisible
- Alle sichtbaren Zellen

Letzte Zelle im benutzten Bereich ermitteln

```
ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Activate
```

Bestimmter Bereich auswählen

```
Range(Selection, Selection.SpecialCells(xlCellTypeLastCell).Select
```

Letzte Zelle in Markierung Ermitteln

```
s = Selection(Selection.Count).Address
```

Dynamischer Datenbezug

```
Dim Mitarbeiterliste As Name
 On Error Resume Next
 ThisWorkbook.Names(”Mitarbeiterliste”).Delete
 ThisWorkbook.Names.Add ”Mitarbeiterliste”, _
   ThisWorkbook.Sheets(”Tabelle6”).Range(”A1”).CurrentRegion
 Range(”Mitarbeiterliste”).Select
```

Bedingte Formatierung einfügen

Mit dem `FormatConditions-Objekt`

Schriftart ändern

Bold
- Fett

Color

- Farbe. Diese Eigenschaft gibt die Primärfarben des Objektes wieder. Möglich sind hierbei folgende Konstanten: `vbBlack`, `vbRed`, `vbGreen`, `vbYellow`, `vbBlue`, `vbMagenta`, `vbCyan` und `vbWhite`.

ColorIndex

- Farbindex. Diese Eigenschaft gibt die Farbe des Rahmens, der Schriftart oder des Innenraums zurück. Es existieren in Excel genau 56 Farben

FontStyle

- Diese Eigenschaft sagt aus, welcher Schriftschnitt verwendet wird. Möglich sind u.a. Fett- und Kursivdruck

Italic

- Kursiv

OutLineFont

- Kontur

Shadow

- Schatten

Strikethrough

- Durchstreichen

Subscript

- Tiefgestellt

Subperscript

- Hochgestellt

Underline

- Unterstrich

Inhalte löschen bei roter Schrift

```
Dim Zelle As Range
For Each Zelle In ActiveSheet.UsedRange
    If Zelle.Font.ColorIndex = 3 Then
        Zelle.ClearContents
    End If
Next Zelle
```

Syntax von Characters

```
Ausdruck.Characters([Start,] [Length])
```

- `Ausdruck` Erforderlich. Ein Ausdruck, der ein Objekt der Liste **Betrifft** zurückgibt.
- **`Start`** Optionaler `Variant`-Wert. Das erste zurückzugebende Zeichen. Falls dieses Argument den Wert 1 hat oder nicht angegeben wird, gibt diese Eigenschaft einen Zeichenbereich zurück, der mit dem ersten Zeichen beginnt.
- **`Length`** Optionaler `Variant`-Wert. Die Anzahl der zurückzugebenden Zeichen. Falls dieses Argument nicht angegeben wird, gibt diese Eigenschaft den Rest der Zeichenfolge zurück (alle Zeichen nach **`Start`**).

Farben mit Indexnummern erstellen

```
Dim i As Integer
Range(”A1”).Select
 For i = 1 To 28
  ActiveCell.Interior.ColorIndex = i
  ActiveCell.Offset(1, 0).Value = i
  ActiveCell.Offset(0, 1).Select
 Next i

 Range(”A4”).Select
```

```
For i = 29 To 56
 ActiveCell.Interior.ColorIndex = i
 ActiveCell.Offset(1, 0).Value = i
 ActiveCell.Offset(0, 1).Select
Next i
```

Gültigkeitsprüfung

...

Bedingte formatierung

...

Eckpositionen der Markierung ermitteln

```
MsgBox "Start Zelle: " & ActiveCell.Address & Chr(10)
  & "Ende Zelle: " & Selection(Selection.Count).Address
```

Markierter Bereich wird zum Druckbereich

```
Dim s As String
S = Selection.Address
ActiveSheet.PageSetup.PrintArea = s
```

- Speichern Sie den Zellbezug der Markierung in einer String-Variablen und übergeben Sie diesen an die Eigenschaft `PrintArea` des Objekts `PageSetup`. Hiermit legen Sie den Druckbereich fest.
- Setzen Sie die Eigenschaft `PrintArea` auf den Wert `False` oder auf die leere Zeichenfolge `("")`, um das gesamte Blatt als Druckbereich festzulegen, also den gerade festgelegten Druckbereich zu löschen.

Zelle mit "X" ausfüllen

- `ActiveCell.Value = "X"`
- `ActiveCell.HorizontalAlignment = xlFill - Alignment = Ausrichten`

Diverses

- Standardformat = `"General"`
- `ActiveCell.Clear` - Alles Löschen
- Empfehlenswert ist hin und wieder die Bildschirmaktualisierung einzuschalten.
- Schonen Sie Ihre Augen und Ihren Bildschirm. Geschwindikeitsoptimierung
- # = Lattenzaun

6.4 Syntaxzusammenfassung

Zugriff auf ausgewählte Bereiche

ActivCell	aktive Zelle
Selection	markierter Bereich oder markiertes Objekt
RangeSelection	markierter Bereich (auch dann, wenn zusätzlich ein anderes Objekt ausgewählt wurde
UsedRange	genutzter Bereich im Tabellenblatt

Auswahl von Bereichen

Range("A3")
Range("A3:B5")

Range(”A3:B5, C7”)	
Range(”Name”)	
Evaluate(”Name”)	
[A3] oder [A3:B5] oder [Name]	Kurzschreibweise für Range. Bzw Evaluate
Range.Offset(z, sp)	liefert um einen z Zeilen und sp Spalten versetzten Bereich
Range.Resize(z, sp)	verändert die Bereichsgrösse auf z Zeilen und sp Spalten
Range.Select	wählt den angegebenen Bereich aus
Range.Activate	wie oben
GoTo Range	wählt den angegebenen Bereich aus
GoTo Range, True	wie oben, zeigt Bereich aber auch an
Union(Range1, Range2, …)	Vereinigung der angeführten Bereiche
Intersect(Range1, Range2, …)	Schnittmenge der angeführten Bereiche

Zugriff auf spezielle Zellen

Range.Cells	Aufzählobjekt aller Zellen
Range.Cells(n)	n-te Zelle (1=A1, 2=B1, 257=A2 etc.)
Range.Cells(z, sp)	Zelle der z-ten Zeile sp-ten Spalte
Range.Areas	Aufzählobjekt aller rechteckigen Bereiche
Range.Areas(n)	n-ter rechteckiger Bereich
Range.EntireColumn	Spalten, in denen sich der Bereich befindet
Range.EntireRow	wie oben für Zeilen
Range.Columns(n)	Zugriff auf einzelne Spalten
Range.Rows(n)	Zugriff auf einzelne Zeilen
Range.SpecialCells(typ)	Zugriff auf leere, sichtbare, untergeordnete etc. Zellen
Range.End(xlDown / xlUp)	Zugriff auf letzte Zelle in einer Richtung
Range.CurrentRegion	Zugriff auf zusammengehörigen Zellbereich
Range.[Direct]Precedents	Zugriff auf Vorgängerzellen (Ausgangsdaten)
Range.[Direct]Dependents	Zugriff auf Nachfolgerzellen (Formeln)
Range.ListHeaderRows	ermittelt die Anzahl der Überschriftszeilen eines Bereichs

Benannte Zellbereiche, Adressen von Bereichen

Names.Add ”test”, ”d5”	definiert den Namen ”Test” mit dem Bezug auf die Zelle D5
[Test].Select	wählt den Zellbereich ”Test” aus
Names(”Test”).RefersTo	liefert Bereichsbezeichnung (z.B. ”=Tabelle1!$F$4:$G$6”)
Names(”Test”).RefersToR1C1	wie oben, aber in R1C1-Schreibweise
Names(”Test”).RefersToR1C1Local	wie oben, aber in Z1S1-Schreibweise
Names(”Test”).Delete	löscht den Namen ”Test”
Range.Adress(…)	liefert Zeichenkette mit Bereichsadresse
Range.AdressLocal(…)	wie oben, aber Z1S1- statt R1C1-Schreibweise

Daten in Zellbereichen einfügen / löschen

Range.ClearContents	Zellinhalte löschen
Range.ClearFormats	Formatierung der Zellen löschen
Range.Clear	Inhalte und Formate löschen
Range.ClearNotes	Notizen löschen

Range.Delet [xlToLeft oder xlUp] Zellen löschen
Range.Insert [xlToRight oder xlDown] Zellen einfügen

Inhalt und Format einzelner Zellen

Range.Value	Wert der Zelle
Range.Text	formatierte Zeichenkette mit Inhalt der Zelle (read-only)
Range.Characters(start, anzahl)	einzelne Zeichen einer Textkonstanten
Range.Formula	Formel der Zelle in A1-Schreibweise, englische Funktionsnamen
Range.FormulaR1C1	Formel in R1C1-Schreibweise, engl. Funktionsnamen
Range.FormulaLocal	Formel der Range A1-Schreibweise, deutsche Funktionsnamen
Range.FormulaR1C1Local	Formel in Z1S1-Schreibweise, deutsche Funktionsnamn
Range.NoteText(Text, Start, End)	liest oder verändert bis zu 255 Zeichen der Notiz zur Zelle
Range.Font	Verweis auf Schriftartobjekt
Range.VerticalAlignment	vertikale Ausrichtung (links / rechts / zentriert / bündig)
Range.HorizontalAlignment	horizontale Ausrichtung (oben / unten / mitte)
Range.Orientation	Textrichtung (horizontal / vertikal)
Range.WrapText	Zeilenumbruch
Range.ColumnWidth	Breite der ganzen Spalte
Range.RowHeight	Höhe der ganzen Zeile
Range.NumberFormat	Zeichenkette mit Zahlenformat
Range.Style	Zeichenkette mit Formatvorlagenname
Range.BorderAround art, stärke	stellt den Gesamtrahmenein
Range.Borders	Verweis auf Rahmenobjekt
Range.Row	Zeilennummer der Zelle
Range.Column	Spaltennummer der Zelle

7 Zeilen und Spalten

7.1 Zeilen und Spalten markieren und ansteuern

Markierung einer einzelnen Zeile bzw. Spalte

- `Rows("3:3").Select`
- `Columns("A:A").Select`

Mehrere Zeilen und Spalten markieren

- `Range("2:2, 3:3, 4:4, 10:10").Select`
- `Range("A:A, C:C, E:E").Select`

Erste freie Zelle in Spalte anspringen

- `Range("A65536").End(xlUp).Offset(1, 0).Select`

Anzahl der verwendeten Zeilen

```
Dim l As Long
l = ActiveSheet.UsedRange.Rows.Count
```

Anzahl der verwendeten Spalten

```
Dim l As Long
l = ActiveSheet.UsedRange.Columns.Count
```

7.2 Zeilenhöhe und Spaltenbreite einstellen

Zeilenhöhe und Spatenbreite einstellen

```
Range("A:E").EntireColumn.ColumnWidth = 15
Range("1:4").EntireRow.RowHeight = 20
```

Zeilenhöhe und Spatenbreite für ganze Tabelle einstellen

```
Cells.Select
With Selection
  .EntireColumn.ColumnWidth = 15
  .EntireRow.RowHeight = 20
End With
```

Spaltenbreite optimal anpassen

```
Worksheets(1).Columns("A:G").AutoFit
```

Dynamische Spaltenanpassung

- Wenn Zelle so aus sieht (da Spalte zu eng): `############`

```
Dim Zelle As Range
  For Each Zelle In ActiveSheet.UsedRange
    If IsNumeric(Zelle.Value) And Left(Zelle.Text, 1) = "#" Then
      Columns(Zelle.Column).AutoFit
    End If
  Next Zelle
```

Zeilenhöhe vergrössern

```
Zeile.RowHeight = Zeile.RowHeight + s
```

7.3 Zeilen einfügen und löschen

Zeile einfügen

```
Range(”A1”).Select
Selection.EntireRow.Insert
```

Zeile löschen

```
Rows(”1:4”).Select
Selection.Delete Shift:=xlUp
```

Zeile löschen bei Bedingung

```
If InStr(1, ActiveCell.Value, ”Alt”) Then Selection.EntireRow.Delete
```

Zeilen löschen gröser als 10 Zeichen

```
If Len(ActiveCell.Value) > 10 Then Selection.EntireRow.Delete
```

Versteckte Zeilen löschen

```
Dim Zeile As Range
For Each Zeile In ActiveSheet.UsedRange.Rows
    If Zeile.Hidden Then
        Zeile.Hidden = False
        Zeile.Delete
    End If
Next Zeile
```

7.4 Spalten einfügen, löschen und bereinigen

Spalte einfügen

```
Range(”A1”).Select
Selection.EntireColumn.Insert
```

Mehrere Spalten einfügen

```
Columns(”C:E”).Select
Selection.Insert Shift:=xlToRight
```

SpalteLöschen

```
Range(”A1”).Select
ActiveCell.EntireColumn.Delete
```

Mehrere Spalten löschen

```
Columns(”B:D”).Select
Selection.Delete Shift:=xlToLeft
```

Zweite und vierte Spalte bereinigen

```
Dim Zelle As Object
ActiveSeet.UsedRange.Select
For Each Zelle In Selection
```

If **Zelle.Column = 2** Or **Zelle.Column = 4** Then Zelle.Value = ””

```
Next Zelle
```

Spaltenbuchstaben ermitteln

```
Dim s As String
Dim sbuch As String
  s = ActiveCell.Address
  sbuch = Mid(s, 2, InStr(2, s, "$") - 2)
  MsgBox "Der Spaltenbuchstabe der aktiven Zelle ist << " & _
    sbuch & " >>"
```

- Um den Spaltenbuchstaben der aktiven Zelle zu ermitteln, speichern Sie die Zellenadresse der aktiven Zelle in einer String-Variablen. Danach zerlegen Sie diese Variable und übertragen den Spaltenbuchstaben in die Variable `sbuch`. Dazu setzen Sie die Funktion `Mid` ein, mit welcher Sie einen Teil aus einen String ermitteln können. Dabei müssen Sie als Argumente angeben, um welchen String es sich dabei handeln soll, sowie die Anzahl der Zeichen, die übertragen werden sollen. Diese Anzahl ermitteln Sie mit Hilfe der Funktion "InStr". Dabei setzen Sie nach dem zweiten Zeichen der Variable "s" auf und durchsuchen den Rest der Variabeln nach dem Absolut-Zeichen `$`, Wenn Sie dieses gefunden haben, brauchen Sie nur noch den Wert `2` zu subtrahieren, um den Buchstaben bzw. die Buchstabenkombination zu bekommen.

7.5 Zeilen ein- und ausblenden

Zeilen ausblenden

- Das folgende Makro durchsucht in Spalte `A`, ob die Einträge in den einzelnen Zellen eingerückt sind. Wenn ja, werden die entsprechenden Zellen ausgeblendet.

```
Range("A4").Select
  Do Until ActiveCell.Value = ""
    If ActiveCell.IndentLevel = 1 Then Rows(ActiveCell.Row).Hidden =
True
    ActiveCell.Offset(1, 0).Select
  Loop
```

- Die `IndentLevel`-Eigenschaft liefert einen ganzzahligen Wert, der die Einzugsebene der Zelle liefert. Zwischen `0` (kein Einzug) und `15` (maximaler Einzug) ist alles möglich.

OnKey

```
Private Sub Workbook_Open()
  Application.OnKey "w", "AusblendenZeilen"
End Sub
```

Zeile ausblenden

```
Selection.EntireRow.Hidden = True
```

Jede zweite Zeile ausblenden

```
For i = 1 To 20 Step 2
  Rows(i).Hidden = True
Next
```

Zeilen einblenden

```
Dim Zeile As Object
```

```
  For Each Zeile in ActiveSheet.UsedRange.Rows
    Zeile.Hidden = False
  Next Zeile
```

7.6 Text auf Spalten verteilen

Text aus einer Spalte in mehreren Spalten darstellen

```
Dim Bereich As Range
Dim Zelle As Range
Dim Leerz As Integer
Dim Vorname As String
Dim Nachname As String
    Set Bereich = Range(”A:A”)
    For Each Zelle In Bereich
        If IsEmpty(Zelle) Then Exit For
          Leerz = InStr(Zelle, ” ”)
          Vorname = Left(Zelle, Leerz - 1)
          Nachname = Mid(Zelle, Leerz + 1)
          Zelle = Vorname
          Zelle.Offset(0, 1) = Nachname
    Next
```

Text aus mehreren Spalten in eine Spalte bringen

```
Dim s As String
  Range(”A1”).Select
  Do Until ActiveCell.Value = ””
    s = ActiveCell.Value & ” ” & ActiveCell.Offset(0, 1).Value
    ActiveCell.Value = s
    ActiveCell.Offset(1, 0).Select
  Loop
  Range(”B:B”).Clear
```

7.7 Autofilter aktiveren bzw. deaktivieren

Autofilter Einschalten

```
If Not ActiveSheet.AutoFilterMode = True Then Range(”A1”).AutoFilter
```

Alle gesetzten Filter entfernen

```
ActiveSheet.AutoFilterMode = False
```

Gefilterte Zeilen in anderes Tabellenblatt übertragen

```
ActiveCell.CurrentRegion.SpecialCells(xlVisible).Copy
Sheets(”Tabelle2”).Activate
ActiveSheet.Paste
```

Fitern von Daten auch bei geschützten Tabellen durchführen

```
ActiveSheet.Protect userinterfaceonly:=True
ActiveSheet.EnableAutoFilter = True
```

- Indem Sie die Eigenschaft `EnableAutoFilter` auf `True` setzen, aktivieren Sie die `AutoFilter`-Pfeile.

7.8 Diverses

Spalten ausblenden

```
Columns(”D:F”).EntireColumn.Hidden = True
```

Zwei Spalten vergleichen und den grösseren Wert in die dritte Spalte schreiben

```
Dim i As Long
Dim i2 As Long
  i = Range(”A1”).CurrentRegion.Rows.Count
  For i2 = 3 To i
    If Cells(i2, 1) > Cells(i2, 2) Then
      Cells(i2, 3).Value = Cells(i2, 1).Value
    Else
      Cells(i2, 3).Value = Cells(i2, 2).Value
    End If
  Next i2
```

Wiederholungszeilen und –spalten definieren

```
With ActiveSheet.PageSetup
   .PrintTitleRows = ”$1:$1”
   .PrintTitleColumns = ”$A:$A”
End With
```

Jede zweite Zeile schattieren

```
Dim i As Integer
For i = 1 To Selection.Rows.Count
  If i Mod 2 = 1 Then Selection.Rows(i).Interior.ColorIndex = 15
Next
```

- Mit Hilfe des Operators `Mod` ermitteln Sie das ganzzahlige Ergebnis der Division der beiden Werte aus der Variablen `i`, die bei jedem Schleifendurchlauf hochgezähl wird, und der Anzahl der Zellen in der Markierung. Entsteht als Ergebnis der Division der Wert `1`, dann wird die Zeile innerhalb der Markierung mit der Hintergrundfarbe Grau belegt.

Zeilennummer der letzten benutzten Zeile im Tabellenblatt

?ActiveSheet.Range(”A1”).Specialcells(xlCellTypeLastCell).Row

8 Tabellenblätter

8.1 Tabellen einfügen

Tabelle einfügen

```
Worksheets.Add
```

Tabelle am Anfang einfügen

```
Worksheet.Add Before:=ActiveWorkbook.Worksheet(1)
```

Tabelle am Ende Einfügen

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

Tabelle einfügen und Namen geben

```
Dim neuBlatt As Worksheet
On Error GoTo fehlermeldung
  Set neuBlatt = Worksheets.Add
  neuBlatt.Name = ”Bericht”
fehlermeldung: Blattname schon vorhanden
```

Fünfzig Tabellenblätter einfügen

```
For i = 1 To 50
  Worksheets.Add
  Worksheets(i).Name = Application.UserName & i
Next
```

Ausgewählte Blätter löschen

```
…
```

8.2 Tabellen benennen

Tabelle nach Tagesdatum benennen

```
On Error Resume Next
Worksheets(”Tabelle2”).Name = Date
```

Tabelle nach formatiertem Datum benennen

```
ActiveSheet.Name = Format(Now, ”mmm dd”)
```

Tabelle nach Zelleninhalt benennen

```
Worksheet(1).Name = Range(”B1”).Value
```

Tabelle nach Anwender und Tagesdatum benennen

```
Worksheets(1).Name = Application.UserName & ”, ” & Date
```

Tabelle einfügen und benennen kombinieren

```
Worksheets.Add.Name = ”Tabelle1”
```

8.3 Tabellen löschen

Tabellenblatt löschen

```
On Error GoTo fehler
  Sheets(”Tabelle1”).Delete
  Exit Sub
fehler:
MsgBox ”Es gibt keine Tabelle1 zum Löschen”
```

Tabelle löschen ohne Rückfrage

```
Application.DisplayAlerts = False
Seets(1).Delete
```

Alle Tabellen löschen, nur die aktive Tabelle nicht

```
Dim Blatt As Object
Application.DisplayAlerts = False
For Each Blatt In Sheets
  If Blatt.Name <> ActiveSheet.Name Then
  Blatt.Delete
End If
Next Blatt
  ChDir (”C:\Eigene Dateien”)
  Application.Dialogs(xlDialogSaveAs).Show ActiveSheet.Name
  Application.DisplayAlerts = True
```

Alle Tabellen löschen bis auf die erste

```
For i = Worksheets.Count To 1 Step -1
  Sheets(i).Delete
Next
```

Alle leeren Tabellen in Arbeitsmappe löschen

```
Application.DisplayAlerts = False
On Error Resume Next
For i = 1 To ActiveWorkbook.Sheets.Count
  Sheets(i).Activate
  If   ActiveCell.SpecialCell(xlLastCell).Address   =   ”$A$1”   Then
Sheets(i).Delete
Next i

ActiveWindow.SelectedSheets.Delete
Application.DisplayAlerts = False
```

8.4 Tabellen markieren

Vorheriges Blatt aktivieren

```
On Error Resume Next
ActiveSheet.Previous.Activate
```

Nächstes Blatt aktiveren

```
On Error Resume Next
ActiveSheet.Next.Activate
```

Mehrere Tabellenblätter markieren

```
On Error Resume Next
Sheets(Array(”Tabelle1”, ”Tabelle2”, ”Tabelle3”)).Select
```

Alle Tabellen markieren

```
Dim l As Long
Dim lTab As Long
Dim TabArray() As Long
    lTab = ThisWorkbook.Worksheets.Count
    ReDim TabArray(1 To lTab)
    On Error Resume Next
    For l = 1 To lTab
        TabArray(l) = l
    Next l
    ThisWorkbook.Worksheets(TabArray).Select
```

- Ermitteln Sie mit der Eigenschaft `Count` die Anzahl der Tabellenblätter, die in der Arbeitsmappe enthalten sind. Mit der Anweisung `ReDim` reservieren Sie Speicherplatz für die dynamische Datenfeldvariable `TabArray`. Danach füllen Sie den Array mit Hilfe einer `For Next`-Schleife. Im Anschluss daran werden alle Tabellenblätter der markiert

8.5 Tabelle schützen

Variante 1

```
Dim i As Integer
For i = 1 To Worksheets.Count
  Sheets(i).Protect ”Passwort” & i
Next
```

Variante 2

```
Dim neuBlatt As Worksheet
  For Each neuBlatt In Worksheets
   neuBlatt.Activate
   neuBlatt.Protect ””
  Next neuBlatt
End Sub
```

Variante 3

```
Dim neuBlatt As Worksheet
Dim i As Integer
For i = 1 To Worksheets.Count
  Set neuBlatt = Worksheets(i)
  neuBlatt.Activate
  neuBlatt.Project ””
Next i
```

Syntax für Tabellenschutz

```
ActiveSheet.Protect(Password, DrawingObjects, Contents, _
  Scenarios, UserInterfaceOnly)
```

8.6 Tabellenblätter ausblenden

Tabellenblatt ausblenden

```
On Error Resume Next
Sheets(”Tabelle1”).Visible = False
```

Tabellenblätter sicher ausblenden

```
Sheets(”Tabelle1”).Visible = xlVeryHidden
```

Tabellen je nach Status ein- oder ausblenden

```
Dim Blatt As Worksheet
On Error Resume Next
For Each Blatt In ActiveWorkbook.Worksheets
    Select Case Blatt.Visible
        Case xlSheetHidden: Blatt.Visible = xlSheetVisible
        Case xlSheetVisible: Blatt.Visible = xlSheetHidden
    End Select
Next Blatt
```

- Je nach Status wird der Eigenschaft `Visible` entweder die Konstante `xlSheetVisible` bzw. `xlSheetHidden` zugewiesen.

Alle versteckten Blätter anzeigen

```
Dim Blatt As Worksheet
 For Each Blatt In Sheets
   Blatt.Visible = True
 Next Blatt
```

Alle Tabellen ausser der aktiven Tabelle ausblenden

```
Dim Blatt As Object
 For Each Blatt In Sheets
   If Blatt.Name <> ActiveSheet.Name Then
     Blatt.Visible = False
   End If
 Next Blatt
```

8.7 Drucken, kopieren, verschieben

Druckbereich setzen

```
ActiveSheet.PageSetup.PrintArea = Selection.Address
```

Druchbereich festlegen

```
Worksheets(”Tabelle1”).PageSetup.PrintArea = ”$A$1:$E$80”
```

Druckbereich nach Verwendung Festlegen

```
On Error Resume Next
Range(”A1”).Select
 ActiveSheet.PageSetup.PrintArea = ActiveCell.CurrentRegion.Address
```

Tabellenblatt drucken

```
Sheets(”Tabelle1”).PrintOut
```

Verwendeter Bereich der Tabelle kopieren

```
ActiveSheet.UsedRange.Copy
```

Markierter Bereich drucken

```
Selection.PrintOut Copies:=1, Collate:=True
```

Mehrere Kopien drucken

```
Sheets("Tabelle1").PrintOut Copies:=2
```

Mehrere Tabellenblätter drucken

```
Sheets(Array("Tabelle4", "Tabelle1", "Tabelle2")).PrintOut
```

Den integrierten Drucken-Dialog aufrufen

```
Application.Dialogs(xlDialogPrint).Show
```

- Alle Dialogs-Konstanten im Objektkatalog unter **`XlBuiltInDialog`**

Wieviele Druckseiten enthält die Tabelle

```
MsgBox "Die Tabelle enthält : " & _
  ExecuteExcel4Macro("Get.Document(50)") & " Druckseite(n)"
```

- Um diese Aufgabe zu lösen, müssen Sie auf ein Excel-4.0-Makro zurückgreifen. Dieses Makro starten Sie mit der Methode `ExecuteExcel4Macro`

Tabellenblatt kopieren

- Es soll eine Kopie des ersten Tabellenblatts ganz rechts in der Arbeitsmappe eingefügt werden.

```
Dim s As String
Dim i As Integer
  s = InputBox("Bitte geben Sie den Namen des Blattes ein!", _
    "Blattnamen vergeben", "Tabelle1")
  If s = "" Then Exit Sub
  i = Sheets.Count
 On Error Resume Next
 Sheets(1).Copy After:=Sheets(i)
 ActiveSheet.Name = s
```

Tabellenblatt verschieben

```
On Error Resume Next
Sheets("Tabelle1").Move After:=Sheets(Sheets.Count)
```

Tabelle in andere Arbeitsmappe kopieren

```
Dim Mappe As Workbook
Dim Blatt As Object
  Set Blatt = ActiveSheet
  Application.ScreenUpdating = False
  On Error Resume Next
  Set Mappe = Workbooks.Open("C:\eigene Dateien\Mappe1.xls")
    Blatt.Copy Before:=Mappe.Sheets(1)
    Mappe.Save
    Mappe.Close
  Application.ScreenUpdating = True
```

8.8 Fusszeile beschriften

Fusszeile mit Anwendernamen

```
ActiveSheet.PageSetup.RightFooter = Application.UserName
```

Fusszeile mit Pfad

```
ActiveSheet.PageSetup.LeftFooter = ActiveWorkbook.Path _
  & "\" & ActiveWorkbook.Name
```

Fusszeile mit Dokumenteigenschaften füllen

```
With ActiveSheet.PageSetup
  .LeftFooter = _
     ActiveSheet.Parent.BuiltinDocumentProperties("Company")
  .RightFooter = _
     ActiveSheet.Parent.BuiltinDocumentProperties("Author")
End With
ActiveWindow.SelectedSheets.PrintPreview
```

8.9 Konsolidieren

Tabellenblätter konsolidieren – Alle untereinander kopieren

```
Dim KBereich As Range
Dim ZBereich As Range
With ActiveWorkbook
  .Worksheets.Add Before:=.Worksheets(1)
  For i = 2 To .Worksheets.Count
    Set KBereich = .Worksheets(i).UsedRange
    Set ZBereich = Worksheets(1).Cells(Rows.Count, "A").End(xlUp)(2)
     'letzte Zeile ermitteln des ersten Blattes
    KBereich.Copy Destination:=ZBereich
  Next
End With
```

Tabellenblätter summieren (Festwert)

```
Dim erg As Long
Dim i As Integer
Dim lTab As Integer
  lTab = ActiveWorkbook.Sheets.Count
  Sheets(lTab).Activate
  For i = 1 To ActiveWorkbook.Sheets.Count - 1
    If  IsNumeric(Sheets(i).Range("B6").Value)  Then  erg  =  erg  +
Sheets(i).Range("B6").Value
  Next i
  Range("B2").Value = erg
```

Tabellenblätter summieren (Verknüpfung)

```
Dim BArray()
Dim i As Integer
Dim lTab As Integer
Dim s As String
  lTab = ThisWorkbook.Worksheets.Count
  Sheets(lTab).Activate
ReDim BArray(1 To lTab)
```

```
  For i = 1 To lTab - 1
    BArray(i) = Sheets(i).Name & "!B6+"
    s = s & BArray(i)
  Next i
  s = "=" & s
  s = Left(s, Len(s) - 1)
  Range("B2").Value = s
```

- Ermitteln Sie das letzte Tabellenblatt der Arbeitsmappe und aktivieren Sie es. Da Sie die Verknüpfungen in der Zielzelle erhalten möchten, müssen Sie die Namen der einzelnen Tabellenblätter sowie die Zelle, die summiert werden soll, in einen Array einlesen. Diesen Array definieren Sie mit der Anweisung "ReDim" und legen ihn in Grösse an, die ausreicht, um alle Tabellennamen in der Arbeitsmappe aufzunehmen. In einer For Next-Schleife füllen Sie den array, inde, Sie jeweils den Namen und die summierende Zelle angeben. Bei Schleifenaustritt haben Sie folgenden Textstring:
- `Januar!B6+Februar!B6+März!B6+April!B6+Mai!B6+Juni!B6+`
- Damit der Textstring als Formel erkannt wird, müssen Sie ein führendes Gleichheitszeichen einfügen sowie das letzte Zeichen + aus dem String entfernen. Dazu nutzen Sie den Verkettungsoperator `&` sowie die Textfunktionen `Left` und `Len`.

8.10 Diverses

Name der Vorgängertabelle ermitteln

```
ActiveSheet.Previous.Name
```

Name der Nachfolgertabelle ermitteln

```
ActiveSheet.Next.Name
```

Aktive Tabelle in neue Datei kopieren

```
ActiveSheet.Copy
```

- Die Eigenschaft ActiveSheet im Zusammenspiel mit der Methode Copy hat als Ergebnis, dass das aktuelle Tabellenblatt automatisch in eine neue Arbeitsmappe kopiert wird.

Tabellenblatt suchen

```
Dim TabBlatt As Worksheet, BlattName As String
  BlattName = InputBox("Bitte geben Sie einen Suchbegriff ein!", _
    "Tabellenblatt suchen")
  For Each TabBlatt In Sheets
    If InStr(LCase(TabBlatt.Name), LCase(BlattName)) > 0 Then
      TabBlatt.Activate
      Exit Sub
    End If
  Next
  MsgBox "Leider kein Tabellenblatt mit diesem Namen.", _
    vbInformation
End If
```

Tabellenblätter alphabetisch sortieren

```
Dim iMax As Integer
Dim Ibl As Integer
Dim ibl2 As Integer
  Application.ScreenUpdating = False
```

```
  iMax = ActiveWorkbook.Worksheets.Count
  For Ibl = 1 To iMax
    For ibl2 = Ibl To iMax
      If UCase(Worksheets(ibl2).Name) < _
        UCase(Worksheets(Ibl).Name) _
      Then
        Worksheets(ibl2).Move before:=Worksheets(Ibl)
      End If
    Next ibl2
  Next Ibl
  Application.ScreenUpdating = True
```

- Um das Sortieren von Arbeitsblättern durchzuführen, müssen Sie zwei verschachtelte For Next-Schleifen durchlaufen. Beide haben als Endbedingung immer die Anzahl der Tabellen, die in der Mappe enthalten sind. Innerhalb der zweiten Schleife werden die Namen der Tabellenblätter verglichen. Beim Vergleich der Tabellennamen werden diese erst einmal in Grossbuchstaben gewandelt, um sicherzustellen, dass die Gross- und Kleinschreibung beim Sortiervorgang keine Rolle spielt. Je nach Vergleichsergebnis werden die einzelnen Tabellen dann innerhalb der Arbeitsmappe mit Hilfe der Methode `Move` verschoben oder nicht

Tabellenblatt als E-Mail versenden

```
Dim s As String
  s = InputBox(”Geben Sie den Empfänger des e-Mails ein!”)
  If s = ”” Then Exit Sub
  ActiveSheet.Copy
  ActiveWorkbook.SaveAs ”Anhang.xls”
  Application.Dialogs(xlDialogSendMail).Show s
```

Bilder in Tabelle automatisch einfügen

```
Dim Picture As Picture
Dim s As String
  s    =    Application.GetOpenFilename(”    Bilder(*.Jpg;    *.Bmp;
*.Gif),*.jpg, *.bmp, *.gif”)
  Range(”A1”).Value = s
  Range(”A2”).Select
  On Error GoTo abbruch
  If s = ”Falsch” Then
    Range(”A1”).Clear
  Else
    Set Picture = ActiveSheet.Pictures.Insert(s)
    Picture.ShapeRange.Height = 220
    Range(”A1”).Select
  End If
abbruch:
```

- Wenden Sie die Methode `GetOpenFilename` an. Sie zeigt das Standarddialogfeld Öffnen an, ohne jedoch irgendwelche Dateien zu öffnen. Die Auswahl der Dateien, die angezeigt werden sollen, können Sie selber festlegen. Im nächsten Schritt erfolgt eine Prüfung, ob der Name der Grafikdatei auch in Zelle A1 geschrieben wurde. Wenn ja, wird die Grafikdatei über die Methode ”Insert” eingefügt. Danach können Sie die Höhe der eingefügten Bilddatei über die Eigenschaft `Height` des Aufliestungsobjekts `ShapeRange` anpassen.

Syntax der Methode CheckSpelling (Rechtschreibprüfung)

```
Ausdruck.CheckSpelling(CustomDictionary, IgnoreUppercase, _
AlwaysSuggest, SpellLang)
```

CustomDictionary

- Gibt das Benutzerwörterbuch an

IgnoreUppercase

- ImmerVorschlagen. Excel zeigt eine Liste mit alternativen Schreibweisen an.

AlwaysSuggest

- Excel ignoriert alle Wörter in grossbuchstaben

SpellLang

- Gibt die Sprache des verwendeten Wörterbuchs an

9 Arbeitsmappen

9.1 Arbeitsmappen und Verknüpfungen

Verknüpfte Mappen als Hyperlink ausgeben

```
Dim Mappe As Workbook
Dim VLink As Variant
Dim i As Integer

Set Mappe = ThisWorkbook
Sheets.Add
Range(”A1”).Select
VLink = Mappe.LinkSources(xlExcelLinks)
If Not IsEmpty(VLink) Then
    For i = 1 To UBound(VLink)
        ActiveCell.Hyperlinks.Add ActiveCell, VLink(i)
        ActiveCell.Offset(1, 0).Select
    Next i
End If
ActiveSheet.Columns(1).AutoFit
```

- Definieren Sie zuerst ein Objekt von Typ ”Workbook”. Dadurch ersparen Sie sich später einiges an Schreibarbeit. Danach definieren Sie ein Datenfeld (Array) vom Datentyp `Variant`, indem Sie die Namen und die Pfade der verknüpften Arbeitsmappen speichern. Mit der Anweisung `Set` sagen Sie aus, dass das Objekt ”Mappe” die momentan aktive Arbeitsmappe repräsentierren soll. Fügen Sie ein neues Tabellenblatt mit der Methode `Add` ein. Füllen Sie jetzt das Datenfeld ”VLink” mit allen Excel-Verknüpfungen. Dazu setzten Sie die Methode `LinkSources` mit der Xlink-Konstanten `xlExcelLinks` ein. Die Methode gibt eine Matrix mit Verknüpfungen einer der Arbeitsmappen zurück. Die Namen in der Matrix entsprechen dabei den Namen der verknüpfen Excel-Arbeitsmappen. Wenn keine Verknüpfung besteht, wird der Wert `Empty` zurückgegeben. Liefert die Methode nicht den Wert `Empty` zurück, dann setztn Sie eine For Next-Schleife ein. Dies Schleife fängt bei der ersten Verknüpfung an und endet bei der letzten Verknüpfung. Innerhalb der Schleife fügen Sie mit Hilfe der Methode `Add` einen Hyperlink ein, der als Ziel den Namen sowie die Pfadangabe der verknüpften Arbeitsmappe beinhaltet. Diese beiden Informationen haben Sie vorher im Datenfeld ”VLink” gespeichert. Danach positionieren Sie eine Zelle weiter nach unten und fügen den nächsten Hyperlink ein.

Verknüpfungen aus der Arbeitsmappe entfernen

```
Dim VLink As Variant
Dim i As Integer
VLink = ActiveWorkbook.LinkSources(xlExcelLinks)

If Not IsEmpty(VLink) Then
 For i = 1 To UBound(VLink)
     ActiveWorkbook.ChangeLink Name:=VLink(i), _
     newname:=ThisWorkbook.Name
 Next i
End If
```

- Definieren Sie ein Datenfeld (Array) vom Datentyp `Variant`, indem Sie die Namen und die Pfade der verknüpften Arbeitsmappen mit Hilfe der Methode `LinkSources`

speichern. Im Anschluss daran fragen Sie ab, ob die Arbeitsmappe überhaupt Verknüpfungen enthält. Wenn nicht, meldet die Funktion `IsEmpty` den Wert `True` zurück, was bedeutet, dass das Datenfeld `VLink` leer ist. Im anderen Fall wird eine For Next-Schleife aufgesetzt. Die Schleife beginnt bei der ersten Verknüpfung und endet bei der letzten Verknüpfung, die im Datenfeld `VLink` steht. Den letzten Eintrag des Datenfelds `VLink` ermitteln sie mit Hilfe der Funktion `UBound`. Innerhalb der Schleife wenden Sie die Methode `ChangeLink` an. Diese Methode ändert eine Verknüpfung von einem Dokument zu einem ander Dokument. Als erstes Argument müssen Sie die Quelle angeben, also die Verknüpfung zur anderen Arbeitsmappe, die Sie aus dem Datenfeld `VLink` herausholen. Das zweite Argument stellt den Namen der neuen Verknüpfung dar. Als neue Verknüpfung geben Sie den Namen der aktiven Arbeitsmapp an.

- Da die Methode `CangeLink` lediglich den Namen der Arbeitmappe austausch, ist bei dieser Art der Entfernung von Verknüpfungen Vorsicht geboten. Die externen Verknüpfungen können auf diese Art natürlich entfernt werden; die Methode belässt aber innerhalb der Verknüpfung die ursprünglichen Zellenbezüge, die dann auf die aktive Arbeitsmappe verweisen.

Verknüpfungen in Festwerte umwandeln

```
Sub ExterneLinksInFestwerteUmsetzen()
Dim Blatt As Worksheet
Dim Zelle As Object
 For Each Blatt In ActiveWorkbook.Worksheets
  Blatt.Activate
  Set Zelle = Cells.Find _
   (what:=”[”, _
   After:=ActiveCell, _
   LookIn:=xlFormulas, _
   lookat:=xlPart, _
   searchorder:=xlByRows, _
   searchdirection:=xlNext, _
   MatchCase:=False)
  While TypeName(Zelle) <> ”Nothing”
      Zelle.Activate
      Zelle.Formula = Zelle.Value
      Set Zelle = Cells.FindNext(After:=ActiveCell)
  Wend
 Next Blatt
End Sub
```

- Bei der Suche nach Verknüpfungen setzen Sie die Methode `Find` ein, der Sie als Suchzeichen die eckige Klammer übergeben. Mit der Funktion `TypeName` können Sie den Status der Variablen `Zelle` abfragen. Entspricht dieser Status dem Wert `Nothing`, kann keine Verknüpfung auf der aktiven Tabelle mehr gefunden werden. Diese haben Sie entweder schon alle umgesetzt bzw. es existieren überhaupt keine Verknüpfungen auf dem Tabellenblatt. Mit Hilfe der Methode `FindNext` setzen Sie die Suche, die mit der Methode `Find` begonnen wurde, mit demselben Suchkriterium fort. Erreicht die Suche das Ende des angegebenen Suchbereichs, beginnt sie erneut am Anfang dieses Bereichs. Da Sie aber direkt nach der Suche die Verknüpfungen in Festwerte umwandeln, liefert die Variable ”Zelle” irgendwann den Wert `Nothing` zurück und Sie können das nächste Tabellenblatt aktivieren und durchsuchen.

Arbeitsmappe als Verknüpfung auf den Desktop legen

```
Sub ArbeitsmappeAufDesktopLegen()
Dim wsh As Object
Dim o_Sh As Object
Dim s_DeskTop As String
  Set wsh = CreateObject(”WScript.Shell”)
  s_DeskTop = wsh.SpecialFolders(”Desktop”)
  Set o_Sh = wsh.CreateShortcut(s_DeskTop & _
    ”\” & ThisWorkbook.Name & ”.lnk”)
  With o_Sh
    .Targetpath = ThisWorkbook.FullName
    .Save
  End With
  Set wsh = Nothing
End Sub
```

- Erstellen Sie zuerst einen Verweis auf das Objekt `Wscript.Shell` mit der Methode `CreateObject`. Danach definieren Sie als Speicherort für Ihre Verknüpfung Ihren Windows Desktop. Dazu setzen Sie die Eigenschaf `SpecialFolders` ein. Jetzt erzeugen Sie mit der Methode `CreateShortcut` eine Verknüpfung auf Ihrem Desktop.
- Achten Sie darauf, dass Sie beim Namen der Verknüpfung die Endung `LNK` oder `URL` angeben, sonst kommt es zu einer Fehlermeldung.
- Möchten Sie mehr zum Objekt von Windows Scripting Host (WSH) erfahren, dann binden Sie in der Entwicklungsumgebung die Objektbibliothek Windows Scripting Host Object Model (Vers. 1.0) ein.

Verknüpfte Mappen öffnen

```
Dim V_Mappen As Variant
Dim i  As Integer
   V_Mappen = ActiveWorkbook.LinkSources(xlExcelLinks)
   If Not IsEmpty(Links) Then
      For i = 1 To UBound(V_Mappen)
         Workbooks.Open V_Mappen(i)
      Next i
   Else
      MsgBox ”In dieser Mappe sind keine Verknüpfungen zu anderen
Mappen enthalten!”
   End If
```

Verknüpfungen farblich kennzeichnen

```
Sub VerknüpfungenKenntlichMachen()
Dim s As String
Dim Zelle As Range
Application.ScreenUpdating = False
With Selection
    For Each Zelle In ActiveSheet.UsedRange
     s = Zelle.Formula
     If s Like ”*.xls*” Then
     Zelle.Interior.ColorIndex = 3
     Else
     Zelle.Interior.ColorIndex = xlNone
   End If
   Next Zelle
End With
```

```
Application.ScreenUpdating = True
End Sub
```

9.2 Benutzerdefinierte Listen

Benutzerdefinierte Liste aus Zellbezügen herstellen

```
Application.AddCustomList ListArray:=Columns(”B:B”)
```

Benutzerdefinierte Liste herstellen

```
Application.AddCustomList Array(”Personal”, ”Hardware”, ”Software”,
”Raum”, ”Energie”,  ”Abschreibung”, ”Umlagen”, ”Leistung”,
”Ergebnis”)
```

Benutzerdefinierte Listen löschen

```
Dim i As Integer
Dim j As Integer
  On Error Resume Next
  For i = 1 To Application.CustomListCount
    Application.DeleteCustomList i
  Next i
```

9.3 Formatvorlagen

Aktuelle Formatierung als Formatvorlage

```
ActiveWorkbook.Styles.Add Name:=”Titelrahmen”, basedon:=ActiveCell
```

- Die Methode `Add` erstellt eine neue Formatvorlage und fügt sie der aktiven Arbeitsmappe hinzu. Dabei werden zwei Argumente angegeben: Das erste Argument `Name` beinhaltet den Namen der Formatvorlage. Das zweite Argument sagt aus, wo die Formatvorlage die Formatierung hernehmen soll.

Formatvorlage erstellen

```
On Error Resume Next
With ActiveWorkbook.Styles.Add(Name:=”Überschrift1”)
    .Font.Name = ”Arial”
    .Font.Size = 25
End With
```

Formatvorlagen löschen

```
Dim Fv As Object
  On Error Resume Next
  For Each Fv In ActiveWorkbook.Styles
    Fv.Delete
  Next Fv
```

- Die Eigenschaft `Style` gibt eine Auflistung aller Formatvorlagen zurück, die die Arbeitsmappe enthält. Über eine For Next-Schleife wreden alle Formatvorlgagen der Arbeitsmappen gelöscht, die Formatvorlage `Standard` kann jedoch nicht gelöscht werden. Die On Error-Anweisung fängt den Versuch, diese Foramtvorlage zu löschen, ab.

Formatvorlagen löschen 2

```
Dim Fv As Object
```

```
For Each Fv In ActiveWorkbook.Styles
  If Fv.BuiltIn = False Then
    Fv.Delete
  End If
Next Fv
```

- Die Formatvorlage `Standard` können Sie aber auch nocht über eine andere Möglichkeit vor dem Versuch der Löschung schützen. Dazu verwenden Sie die Eigenschaft `BuiltIn`. Die Eigenschaft `BuiltIn` meldet den Wert `True`, wenn die Formatvorlage integriert, also fest in Excel eingebaut ist.

9.4 Diverses

Geöffnete Arbeitsmappen zählen

```
MsgBox "Es sind zur Zeit " & Application.Workbooks.Count & _
  " Datei(en) geöffnet", vbInformation
```

Arbeitsmappe einfügen und Anzahl Tabellen einstellen

```
Application.SheetsInNewWorkbook = 1
Workbooks.Add
```

Arbeitsmappe drucken

```
ActiveWorkbook.PrintOut
```

Neue Mappe erstellen

```
Dim neuMappe As Workbooks
Set neuMappe = Workbooks.Add
With neuMappe
  .Methoden
  .Eigenschaften
End With
```

Das Speichern einer Arbeitsmappe

```
ActiveWorkbook.SaveAs Filename:="MeineMakros"
```

Öffnen einer Arbeitsmappe

```
Dim wkbNeu As Workbook
Set wkbNeu = Application.Workbooks.Open(FileName:= _
 "C:\Eigene Dateien\MeineMakros.xls")
```

Kopfzeile mit komplettem Pfad

```
Sub KopfzeileMitPfadangabe()
 ActiveSheet.PageSetup.RightHeader = ActiveWorkbook.FullName
 ActiveSheet.PrintPreview
 'ActiveWindow.SelectedSheets.PrintPreview
End Sub
```

Eigenschaft	Anordnung
`LeftFooter`	Fusszeile links
`CenterFooter`	Fusszeile Mitte
`RightFooter`	Fusszeile rechts
`LeftHeader`	Kopfzeile links

`CenterHeader`	Kopfzeile Mitte
`RightHeader`	Kopfzeile rechts

- Über die Methode `PrintPreview` gehen Sie direkt in die Seitenansicht der Tabelle. Möchten Sie mehrere Tabellen nacheinander in der Seitenansicht betrachten, verwenden Sie den Befehl:

```
ActiveWindow.SelectedSheets.PrintPreview
```

Diverses

- `neuGitter = ActiveWindow.DisplayGridlines`
  `ActiveWindow.DisplayGridlines = Not neuGitter`
- `ActiveWindow.WindowState = xlMinimized / xlMaximized / xlNormal`
- `Application.Caption = ”Die Tabellenkalkulation”`

ActivatePrevious, ActivateNext

- Aktiviert das in der Fensterliste letzte bzw. nächstfolgende Fenster

Programmieren des Anwendungsfensters

```
Sub AnwendungsFensterGrösse()
  Application.WindowState = xlNormal
  Application.Left = 40
  Application.Top = 40
  Application.Height = 400 ‘Höhe
  Application.Width = 600 ‘Breite
End Sub
```

Objekt.Methode

```
Workbooks(Mappenname).Activate
Workbooks(”Demo.xls”).Activate
```

9.5 Syntaxzusammenfassung

Zugiff auf Arbeitsmappen, Fenster und Blätter

Workbooks	Zugriff auf alle Arbeitsmappen
Windows	Zugriff auf alle Fenster
Sheets	Zugriff alle Blätter einer Mappe
SelectedSheets	Zugriff auf Blattguppen (bei Mehrfachauswahl)
Worksheets	Zugriff nur auf Tabellenblätter
Chart	Zugriff nur auf Diagrammblätter
DialogSheet	Zugriff nur auf Dialogblätter
Modules	Zugriff nur auf Modulblätter
Excel4MacroSheets	Zugriff nur auf Excel-4-Makros
Excel4IntlMacroSheets	Zugriff auf internationale Makroblätter
ActiveWorkbook	zur Zeit aktive Arbeitsmappe
ThisWorkbook	Arbeitsmappe, in der sich der Code befindet
ActiveWindow	aktives Fenster
ActiveSheet	aktives Blatt von Fenster / Mappe / Anwendung
ActiveChart	aktives Diagramm von Fenster / Mappe / Anwendung
ActiveDialog	aktiver Dialog von Fenster / Mappe / Anwendung

Umgang mit Arbeitsmappen

Workbook.Activate	bestimmt die aktive Arbeitsmappe
Workbooks.Add	erstellt eine neue leere Arbeitsmappe
Workbook.Cose	schliesst die Arbeitsmappe
Workbook.Open ”Dateiname”	lädt die angegebene Datei
Workbook.Save	speichert die Arbeitsmappe
Workbook.SaveAs ”Dateiname”	wie oben, aber unter dem angegebenen Namen
Workbook.SaveCopyAs ”dn”	wie oben, ohne Namen der Arbeitsmappe zu ändern
Workbook.Name	enthält den Dateinamen ohne Pfad
Workbook.Path	nur Pfad
Workbook.FullName	Pfad plus Dateiname
Workbook.Saved	gibt an, ob Arbeitsmappe gespeichert ist
ApplicationGetOpenFilename	vorhandenen Dateinamen auswählen
ApplicationGetSaveAsFilename	neuen Dateinamen auswählen

Umgang mit Fenstern

Win.Activate	aktiviert das angegebene Fenster
Win.ActivatePrevious	aktiviert das zuletzt aktive Fenster
Win.ActivateNext	aktiviert das nächste Fenster der Fensterliste
Win.Close	schliesst das angegebene Fenster
Win.NewWindow	erzeugt ein neues Fenster
Win.WindowState	xlMaximized, xlMinimized, xlNormal
Win.Visible	ein- / ausgeblendet (True/False)
Win.Caption	gibt den Fenstertitel an
Win.DisplayGridlines	Gitter anzeigen (True/False)
Win.DisplayHeadings	Zeilen und Spaltenköpfe anzeigen (True/False)
Win.Zoom	Zoomfaktor (10-400)
Win.ScollColumn	sichtbare Spaltennummer am linken Rand
Win.ScrollRow	sichtbare Zeilennummer am oberen Rand
Win.Split	gibt an, ob Fenster geteilt ist (True/False)
Win.FreezePane	gibt an, ob Fensterteilung fixiert ist
Win.SplitRow	bestimmt Zeilenanzahl im oberen Fensterteil
Win.SplitColumn	bestimmt Spaltenanzahl im linken Fensterteil
Win.Width/Height	Aussenmasse in Punkt (0.35 mm)
Win.UsableWidht/UsableHeight	Innenmasse in Punkt
Win.Left, Win.Top	Position in Punkt

Umgang mit Fensterausschnitten

Win.Panes	Zugriff auf alle Ausschnitte des Fensters
Win.ActivePane	Zugriff auf den aktiven Ausschnitt des Fensters
pane.Activate	bestimmt den aktiven Ausschnitt
pane.SplitColumn	Zeilennummer am oberen Rand
pane.SplitRow	Spaltennummer am linken Rand

Umgang mit Arbeitsblätter

Sheet.Activate	wählt ein Blatt aus
Sheet.Select False	Mehrfachauswahl
Workbook.Add	fügt ein leeres Tabellenblatt hinzu
Workbook.Add before:=, typ:=	wie oben, plus Position- und Blattyp

Sheet.Copy	kopiert das Blatt in eine neue Mappe
Sheet.Copy Sheet2	kopiert Blatt1 und fügt es vor Blatt2 ein
Sheet.Delete	löscht das Blatt (mit Sicherheitsabfrage)
Sheet.Name	Name des Blatts
Sheet.Visible	ein- oder ausgeblendet

10 Mit dem Dateisystem programmieren

- Kommandos zur Manipulation von Dateien stammen aus drei unterschiedlichen Bibliotheken:
- Die Microsoft Scripting Library ermöglicht mit den File System Objects (FSO) einen objektorientierten Zugang auf Dateien, Verzeichnisse und Textdateien.
- Die in VBA integrierten Kommandos, mit denen nicht nur Text-, sondern auch Binärdateien bearbeitet werden können.
- Schliesslich gibt es noch eine Reihe von Methoden und Eigenschaften, die Excelspezifisch sind und daher zur Excel-Bibliothek zählen. Dazu gehören auch die Funktionen zum Import von Textdateien.

10.1 VBA-Kommandos

Dateien suchen

- NewSearch
- Setzt alle Suchkriterien zurück und leitet eine neue Suche ein

- FileName
- Ermittelt oder bestimmt das Suchkritierium (Dateiname oder Suchmuster oder Pfadname

- FileTyp
- Legt einen bestimmten Dokumententyp fest, z.B. Word-Dokumente. Eine Dateiendung in der Eigenschaft FileName hat dabei Vorrang

- SearchSubFolders
- Legt fest, ob Unterverzeichnisse in die Suche eingeschlossen werden

- LookIn
- Ermittelt oder bestimmt das Startverzeichnis für die Suche. Fehlt die Angabe, wird das aktuelle Verzeichnis angenommen

- Execute
- Führt die Dateisuche aus und liefert die Anzahl der gefundenen Dateien zurück bzw. Null, wenn die Suche nicht erfolgreich verlief

- FoundFiles
- Liefert eine Liste der gefundenen Dateien

Beispiel: Datei suchen

```
Dim Datei As Variant
Dim Meldung As String
With Application.FileSearch
  .NewSearch
  .LookIn = ”C\Eigene Dateien”
  .SearchSubFolders = True
  .FileName = ”*.doc”
  If .Execute > 0 Then
    Meldung = ”Es wurden ” & .FoundFiles.Count & _
      ” Dateien gefunden.” & vbCrLf
    For Each Datei In .FoundFiles
      Meldung = Meldung & Datei & vbCrLf
    Next Datei
  End If
End With
```

Mit dem Dateisystem programmieren

- ChDir
- Wechselt in ein Verzeichnis

- CurDir
- Ermittelt das aktuelle Verzeichnis

- MkDir
- Erstellt ein neues Verzeichnis

- RmDir Pfad
- Das Verzeichnis löschen

- Kill Pfadname
- Löscht eine Datei
- `Kill "C:\test.xls"`

- FileCopy
- Kopiert eine Datei

- FileLen
- Bestimmt die Dateigrösse

- FileDateTime
- Ermittelt Dateidatum und -zeit

- Save
- Speichert die Arbeitsmappe unter ihrem bisherigen Namen

- SaveAs
- Hier muss ein gültiger Dateinamen angegeben werden

- SaveCopyAs
- Wie oben, allerdings ändert sich der Dateinamen der Arbeitsmappe nicht

- GetOpenFilename
- Die Methode zeigt den Dialog zur Dateiauswähl an. Wenn ein gültiger Dateiname ausgewählt wird, gibt die Methode diesen zurück, andernfalls den Wahrheitswert *False*. Die ausgewählte Datei wird aber in keinem Fall geöffnet. Die Methode muss auf das *Application*-Objekt angewendet werden.

- GetSaveAsFilename
- Wie oben, es darf aber auch eine noch nicht existierende Datei angegeben werden.

- Name
- Name der Datei

- Path
- Pfad der Datei

- FullName
- Pfad und Name der Datei

Beispiel: Datei Info

```
Dim Datei As String, Info As String
Datei = "daten.mdb"
  FileSystem.ChDir ("C:\Eigene Dateien")
  Info = Round(FileSystem.FileLen(Datei), 2) / 1024 & " KB " &
     vbCrLf & _
    FileSystem.FileDateTime(Datei)
  MsgBox Info, vbOKOnly + vbInformation, Datei
```

Verzeichnis zurückgeben

- neuPfad = Application.ActiveWorkbook.Path

VBA kennt drei Arten von Dateien

- sequentiolle Dateien
- Binärdateien
- Direktzugriffsdateien

▪ Bei der sequentiellen Speicherung befinden sich alle Daten unmittelbar nebeneinander, die Datei ist also extrem komprimiert und so klein wie möglich. VBA weis nicht, welche Adressen vor der dritten gespeichert sind und wie lange diese Adressen sind. Um an die dritte Adresse heranzukommen, sind daher drei Leseanweisungen nötig.

10.2 Arbeitsmappe Speichern und sichern

Datei Speichern

```
Dim s As String
Const Lw = ”c:\”
Const Pfad = ”c:\eigene Dateien”
    s = ActiveWorkbook.Name
    ChDrive Lw
    ChDir Pfad
    ActiveWorkbook.SaveAs _
     Filename:=s, _
     FileFormat:=xlNormal, _
     Password:=””, _
     WriteResPassword:=””, _
          ReadOnlyRecommended:=False, _
     CreateBackup:=True
```

Syntax der SaveAs-Methode

▪ ActiveWorkbook.SaveAs(Filename, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AddToMru, TextCodePage, TextVisualLayout

▪ Das Argument `FileName` haben sie vorher in der Vakriablen *s* ermittelt. Beim Argument `FileFormat` können sie angeben, in welchem Format Sie Ihre Mappe speichern möchten. Mit dem nächsten Argument `Passwort` können Sie ein Passwort festlegen, welches der Anwender beim Öffnen der Datei eingeben muss, um die Datei laden zu können. Im letzten Beispiel wurde eine Leerzeichenfolge übergeben, was bedeutet, dass kein Passwort beim Öffnen der Datei abgefragt wird. Das Argument `WriteResPassword` sorgt dafür, dass ein Kennwort für die Schreiberlaubnis der Datei eingegeben werden muss. Das Argument `ReadOnlyRecommended` setzen Sie auf `True`, wenn beim Öffnen der Datei in einer Meldung empfohlen werden soll, die Datei mit Nur-Lesen-Zugriff zu öffnen. Belassen Sie das Argument auf dem Wert `False`, unterbleibt Diese Meldung. Das Argument `CreateBackup` setztn Sie auf den Wert `Treu`, wenn Excel von der Mappe eine Sicherungskopie anlegen soll. Excel legt dann eine Sicherungskopie unter demselben Namen mit dem Zusatz `Sicherungskopie von …` und der Endung `wlk` an. Die übrigen Argumente sind in der Praxis nicht so relevant und werden an dieser Stelle nicht weiter beschrieben. Eine Beschreibung können Sie aber in der Online-Hilfe einsehen.

Arbeitsmappe doppelt sichern

```
Dim s As String
Const Lw = ”c:\”
Const Lw2 = ”q:\”
Const Pfad = ”c:\Eigene Dateien”
```

```
    Const Pfad2 = ”q:\Sicherungen”

    s = ActiveWorkbook.Name
    ChDrive Lw
    ChDir Pfad
    ActiveWorkbook.SaveAs Filename:=s, FileFormat:= _
        xlNormal, Password:=””, WriteResPassword:=””, _
        ReadOnlyRecommended:=False, CreateBackup:=True

    ChDrive Lw2
    ChDir Pfad2
    ActiveWorkbook.SaveAs Filename:=s, FileFormat:= _
        xlNormal, Password:=””, WriteResPassword:=””, _
        ReadOnlyRecommended:=False, CreateBackup:=True
```

Integrierter Speichern unter-Dialog aufrufen

```
Application.Dialogs(xlDialogSaveAs).Show
```

Die Ermittlung des Pfades der geladenen Arbeitsmappe

```
Dim s As String
 s = Application.ActiveWorkbook.Path
 MsgBox s
```

Arbeitsmappe gespeichert oder nicht

```
If ActiveWorkbook.Saved = False Then
    MsgBox ”Die Mappe wurde geändert!”
 End If
```

Die Eigenschaft `Saved` liefert den Wert `True`, wenn die aktive Arbeitsmappe seit der letzten Speicherung nicht mehr geändert wurde.

Arbeitsmappe ohne Makros speichern

```
Dim Original As String
Dim Kopie As String
Dim i  As Integer

Original = ActiveWorkbook.Name
Kopie = ActiveWorkbook.Name & ”_Kopie.xls”

Workbooks.Add
ActiveWorkbook.SaveAs Filename:=”C:\Eigene Dateien\” & Kopie

Workbooks(Original).Activate
 For i = 1 To Sheets.Count
  Sheets(i).Copy After:=Workbooks(Kopie).Sheets(i)
 Next i
```

10.3 Arbeitsmappe öffnen

Date nur öffnen, wenn sie nicht schon geöffet ist (Notify:=False)

Workbooks.Open:= Pfad, Notify:=False

- Hat die Arbeitsmappe schon ein anderer Anwender geöffnet, wird die Mappe nicht geöffnet.

Arbeitsmappe öffnen

```
Const Lw = ”c:\”
Const Pfad = ”c:\Eigene Dateien”
Const Datei = ”Mappe2.xls”

    ChDrive Lw
    ChDir Pfad
    On Error Resume Next
    Workbooks.Open Datei
```

Die Syntax der Methode Open

- `Workbooks.Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMRU)`
- Besonders wichtig ist das Argument `UpdateLinks`. Sicher haben sie auch schon einmal beim Öffnen einer Arbeitsmappe die Meldung erhalten, ob Sie die Verknüpfungen in der Arbeitsmappe aktualisieren möchten oder nicht. Diese Abfrage können Sie unterdrücken, indem Sie ein entsprechendes Argument 0-3 einsetzten. Die Bedeutung der verschiedenen Werte entnehmen Sie der folgenden Tabelle:

0	Keine Aktualisierung von Bezügen
1	Aktualisierung von externen Bezügen, jedoch nicht von Fernbezügen
2	Aktualisierung von Fernbezügen, jedoch nicht von externen Bezügen
3	Aktualisierung von externen Bezügen und Fernbezügen

```
Workbooks.Open Filename:=”Mappe2.xls”, UpdateLinks:=0
```

- Verknüpfung immer updaten:
- Setzen Sie die Eigenschaft `AskToUpdateLinks` auf den Wert `True`, um die Aktualisierungsabfrage beim Öffnen von Arbeitsmappen wieder anzuzeigen.

```
Application.AskToUpdateLinks = False
```

Siehe auch Dialoge zur Dateiauswahl

auf Seite 139

Den integrierten Öffnen-Dialog aufrufen

```
Application.Dialogs(xlDialogOpen).Show ”C:\Eigene Dateien”
```

Anzahl der angezeigten Arbeitsmappen im Menü Datei festlegen

```
With Application
  .DisplayRecentFiles = True
  .RecentFiles.Maximum = 9
End With
```

- Mit der Eigenschaft `DisplayRecentFiles` bestimmen Sie, ob die Wiedervorlage-Liste im Menü Datei angezeigt werden soll.

Mehrere Arbeitsmappen öffnen

```
Dim l As Long
Dim Mappen As Variant
```

```
l = 0
Mappen = Application.GetOpenFilename(MultiSelect:=True)
If IsArray(Mappen) Then
   For l = LBound(Mappen) To UBound(Mappen)
      Workbooks.Open Mappen(l)
   Next l
Else
   Workbooks.Open Mappen
End If
```

Syntax der GetOpenFilename-Methode

```
GetOpenFilename(FileFilter, FileIndex, Title, MultiSelect)
```

- Im ersten Argument können Sie bestimmen, welche Dateien überhaupt angezeigt werden sollen. Wenn Sie dieses Argument weglassen, werden alle Dateiein im Dialog angezeigt. Das Argument `FileIndex` gibt die Indexnummer der Standarddatei-Filterkriterien an. Das Argument `Title` bestimmt die Überschrift im Dialog. Hiermit können Sie dem Öffnen-Dialog eine eigene Überschrift zuweisen. Wird das Argument weggelassen, wird standardmässig der Text ”Öffnen” als Dialogüberschrift gewählt. Das Argument `MultiSelect` gibt an, ob es möglich ist im Öffnen-Dialog gleich mehrere Dateien zu markieren.

- Im nächsten Schritt müssen Sie prüfen, wie viele und vor allem welche Dateien der Anwender im Dialog Öffnen markiert hat. Dazu überprüfen Sie mit der Funktion ***IsArray***, ob die Variable ”Mappen” gefüllt ist. Wenn ja, dann ist mehr als eine Date markiert und die For Next-Schleife kann aufgesetzt werden. Diese arbeitet den Array von links nach rechts ab. Es werden alle Dateien, die im Array stehen, mit der Hilfe der Methode `Open` geöffnet. Für den Fall, dass nur eine einzige Datei markiert wurde, meldet die Funktion `IsArray` den Wert `False`. In diesem Fall können Sie die Methode `Open` sofort ausführen und die Variabel ”Mappen” ohne Index anhängen.

Die aktuellste Arbeitsmappe im Verzeichnis öffnen

- Die Funktion `FileDateTime` ermittelt das letzte Änderungsdatum.

```
Dim Dat As Date
Dim DatVgl As Date
Dim Mappe As String
Dim ZMappe As String
Const s = ”c:\Eigene Dateien”

Mappe = Dir(s & ”\*.xls”)
Dat = FileDateTime(s & ”\” & Mappe)
Do Until Mappe = ””
    DatVgl = FileDateTime(s & ”\” & Mappe)
    If DatVgl > Dat Then
    Dat = DatVgl
    ZMappe = s & ”\” & Mappe
    End If
    Mappe = Dir()
Loop
Workbooks.Open Filename:=ZMappe
```

- Mit der Funktion `Dir` können Sie ein Verzeichnis, ja sogar ganze Laufwerke nahc einer bestimmten Datei durchsuchen lassen. Der Funktion übergeben Sie den kompleten Pfad, den Sie in der Konstanten definiert haben, sowie den Dateifilter `(*.xls)`. Das Ergebnis der Funktion speichern Sie in der String-Variablen ”Mappe”. Die Funktion findet nun die erste Datei, die dem Dateifilter entspricht. Ermitteln Sie daraufhin das letzte Änderungsdatum dieser Datei, setzen Sie die Funktion `FileDateTime` ein und speichern Sie das Datum in der Variablen `Dat`. In einer `Do Until`-Schleife erstellen Sie zu Beginn der Schleife eine zweite Varaible `DatVgl`, in der Sie jeweils das letzte Änderungsdatum der nächsten Dateien speichern. Im direkten Vergleich der Variablen ”Dat” mit der Variablen `DatVgl` ermitteln Sie die aktuellste datei und speichern diese in der Variablen `ZName`. Wenden Sie die Funktion `Dir` erneut an, um auf die nächste Datei zu positionieren. Das Ende der Schleife ist erreicht, wenn die Funktion `Dir` einen leeren Wert zurückmeldet. In der Variablen ”Znamen” steht nun die aktuellste Datei aus dem Verzeichnis, die Sie über die Methode `Open` öffnen können.

Prüfen, ob keine Datei ausgewählt wurde

```
Dim v As Variant
   v = Application.GetOpenFilename
    If v = False Then Exit Sub
    Workbooks.Open v
```

10.4 Arbeitsmappe schliessen

Arbeitsmappe schliessen – Änderungen akzeptieren

```
With ActiveWorkbook
  .Sheets(1).Range(”A1”).Value = _
    ”letzte Änderung ” & Now & ” vom Anwender ” &
Application.UserName
  .Close SaveChanges:=True
 End With
 Application.DisplayAlerts = False
```

Arbeitsmappe schliessen – Änderungen verwerfen

```
With Application
  .DisplayAlerts = False
  .DisplayStatusBar = True
  .StatusBar = ”Änderungen an der Datei ” _
     & ActiveWorkbook.Name & _
     ” werden nicht gespeichert!”
  .Wait (Now + TimeValue(”0:00:05”))
  .StatusBar = False
 End With
 ActiveWorkbook.Close
```

Arbeitsmappe schliessen nach zwei Minuten

```
Dim l As Long
Const Puffer As Long = 120

l = Timer
    Do While Timer < l + Puffer
        DoEvents
    Loop
   ActiveWorkbook.Save
```

```
ActiveWorkbook.Close
'Application.Quit
```

- Definieren Sie zuerst einmal die Zeitdauer, nachdem Excel eine unberührte Arbeitsmappe schliessen soll. Die Angabe nehmen Sie in Sekunden vor. Speichern Sid danach einen aktuellen Zeitwert in der Variablen `l` mit Hilfe der Funktion `Timer`. Die Funktion `Timer` gibt einen Wert vom Typ `Single` zurück, der die Anzahl der seit Mitternach vergangenen Sekunden angibt. Diese Zeitangabe können Sie nutzen, um die Dauer abzufragen, die seit dem Start des Makros vergangen ist. In einer `Do While`-Schleife kontrollieren Sie, wann die Zeitdauer aus der Variablen "Puffer" überschritten ist. Danach speichern und schliessen Sie die aktive Arbeitsmappe.

Alle Arbeitsmappen bis auf die aktive schliessen

```
Dim Mappe As Workbook
For Each Mappe In Application.Workbooks
  If Mappe.Name <> ThisWorkbook.Name Then Mappe.Close
Next
```

10.5 Arbeitsmappe löschen

Arbeitsmappe löschen

```
Const Lw = "c:\"
Const Pfad = "c:\eigene Dateien"
Const Datei = "Mappe2.xls"
  On Error GoTo fehler:
  Kill Datei
  MsgBox "Arbeitsmappe " & Datei & " wurde gelöscht!"
  Exit Sub
fehler:
  MsgBox "Es konnte keine Datei mit dem Namen " & Datei & _
    " gefunden werden!"
```

Arbeitsmappe nach Verfallsdatum löschen

```
If Date > CDate("31. Dezember, 2000") Then _
Kill "C:\Eigene Dateien\Geheim.xls"
```

Verzeichnis putzen

```
On Error GoTo ende
  ChDir "C:\Temp"
  Kill "*.*"
ende:
```

10.6 Syntaxzusammenfassung zu VBA-Kommandos

In den Syntaxboxen steht *n* für Dateinamen (etwa "test.dat") und *k* für Kanalnummern

Datei- und Verzeichnisverwaltung

CurDir	liefert das aktuelle Verzeichnis
Environ("Temp")	liefert das Verzeichnis für temporäre Dateien
ChDir n	ändert das aktuelle Verzeichnis
ChDrive drv	ändert das aktuelle Laufwerk
MkDir n	leg ein neues Verzeichnis an
RmDir n	löscht ein leeres Verzeichnis

Name n1 As n2	gibt n1, den neuen Namen n2
FileCopy n1, n2	kopiert n1 nach n2
Kill n	löscht die angegebene(n) Datei(en)
Dir(n [,attribute])	liefert die erste Date, die dem Suchmuster entspricht
Dir	liefert die nächste Datei oder eine leere Zeichenkette
FileLen(n)	liefert die Länge von n in Byte
FileDateTime(n)	liefert Datum und Zeit der letzten Änderung
GetAttr(n)	liefert die Attribute (Read-Only etc.) von n
SetAttr(n, attr	verändert die Attribute von n

Datenkanal öffnen

f = FreeFile	ermittelt freie Datenkanalnummer Datenkanal öffnen, um eine
Open d For Input As #f	… Textdatei zu lesen
Open d For Output As #f	… Textdatei zu schreiben
Open d For Append As #f	… Textdatei zu lesen uns zu schreiben
Open d For Binary As #f	…Binärdatei zu lesen und zu schreiben
Open d For Binary Access Read As #f	… Binärdatei nur zu lesen
Open d For Binary Access Write As #f	… Binärdatei nur zu schreiben
Open d For Random As #f Len=l	… Random-Access-Datei zu lesen und zu schreiben

Dateien via Datenkanal bearbeiten

Close #f	Datenkankal schliessen
Reset	alle offenen Datenkanäle schliessen
EOF(n)	Dateiende erreicht?
LOF(n)	Dateigrösse ermitteln
Loc(n)	aktuelle Position des Datezeigers ermitteln
Seek #f, position	Dateizeiger verändern
Print #f, var1, var2	Zeile im Textformat schreiben
Write #f, var1, var2	wie oben, aber mit Formatzeichen ” und ,
Input #f, var1, var2	einzelne Variablen lesen
Line Input #f, var	ganze Zeile lesen
var = Input(n, #f)	n Zeichen lesen
var = InputB(n, #l)	n Byte lesen
Put #f, , var	Variable / Feld / etc. binär speichern
Get #f, , var	Variable binär lesen

10.7 Excel-spezifische Methoden und Eigenschaften

Laufwerk und Verzeichnisse

ActiveWorkbook.Path	Pfad der aktiven Arbeitsmappe
ActiveWorkbook.Name	Dateiname der aktiven Arbeitsmappe
Application.Path	Pfad zu *Excel.exe*
Application.LibraryPath	Pfad zum globalen *Makro*-Verzeichnis
Application.UserLibraryPath	Pfad zum persönlichen Add-In-Verzeichnis
Application.StartupPath	Pfad zum persönlichen *xlstart*-Verzeichnis
Application.TemplatesPath	Pfad zum persönlichen Vorlagenverzeichnis
Application.AltStartupPath	Pfad zum zusätzlichen Vorlagenverzeichnis

Dateiauswahl

Application.GetOpenFilename	Dateiauswahl (Datei öffnen, nur existierende Dateien)
Application.GetOpenFilename	Dateiauswahl (Datei speichern, mit Sicherheitsabfrage)

Import / Export

Workbooks.OpenText	Textdatei importieren, Variante 1
Workbooks(…).QueryTables.Add	Textdati importieren, Variante 2
Workbooks(…).SaveAs	Tabellenblatt in diversen Formaten speichern

10.8 FileSystemObjekt

Wesentlicher Vorteil von FileSystemObjekt

- Moderner, übersichtlicher und objektorientierter Zugang zu den meisten Funktionen, die zur Analyse des Dateisystems und zum Lesen und Schreiben von Dateien erforderlich sind.
- Sie müssen einen Verweis auf die MS-Scripting-Runtime-Bibliothek machen.

FSO-Bibliothek verwenden

- An der Spitze steht das `FileSystemObject`. Es ist sinnvoll, eine FileSystemObject-Variable global mit `Dim As New` zu definieren.

```
Public fso As New FileSystemObject
```

- Ausgehend von `fso` können Sie nun neue Objekte erzeugen. Die beiden folgenden Kommandos erzeugen z.B. ein `Folder`-Objekt, das auf das existierende Wurzelverzeichnis C: verweist

```
Dim f As Folder
Set f = fso.GetFolder("C:\")
```

- Jetzt können Sie mit `f.Files` auf alle Dateien in diesem Verzeichnis zugreifen, mit `f.SubFolders` auf Verzeichnisse etc. Über Eigenschaften wie `Attributes`, `Name`, `Path` und `Size` etc. können Sie diverse Merkmale der so angesprochenen Dateien und Verzeichnisse ermitteln.

- Im Gegensatz zu den meisten anderen Aufzählungen ist bei `Drives`, `Files`, und `Folders` der Zugriff auf einzelne Elemente durch `Files(n)` nicht möglich! Als Index kann nur der Name des jeweiligen Objekts verwendet werden. Da dieser im Regelfall nicht im voraus bekannt ist, *müssen* Sie mit `For-Each`-Schleifen arbeiten.

- Methoden zum Erzeugen oder Verändern neuer Verzeichnisse und Dateien sind direkt dem `FileSystemObject` untergeordnet, z.B. `CopyFile`, `CreateFolder`, `DeleteFile` etc.

FileSystemObject – Objekthierarchie

FileSystemObject	Spitze der Objekthierarchie
Drives	Aufzählung der Laufwerke und Partitionen
Drive	`Drive`-Objekt zur Bearbeitung des Laufwerks

- Drive – Objekthierarchie

Drive	`Drive`-Objekt
RootFolder	verweist auf Wurzelverzeichnis des Laufwerks
Folder	`Folder`-Objekt

- Folder – Objekthierarchie

Folder	`Folder`-Objekt
Drives	`Drive`-Objekte
Files	Aufzählung aller Dateien im Verzeichnis
File	`File`-Objekt mit den Attributen einer Datei

ParentFolder	übergeordnetes Verzeichnis
Folder	`Folder`-Objekt des übergeordneten Verzeichnisses
SubFolder	`Folders`-Aufzählung
Folder	`Folder`-Objekt mit den Attributen des Unterverzeichnisses

- File – Objekthierarchie

File	`File`-Objekt
ParentFolder	übergeordnetes Verzeichnis
Folder	`Folder`-Objekt des übergeordneten Verzeichnisses

Laufwerke, Verzeichnisse und Dateien

- Eine Liste aller verfügbaren Laufwerke kann leicht über die Aufzählung `fso.Drives` ermittelt werden. Die Eigenschaften der dazugehörigen *Drive*-Objekte geben Aufschluss über die Merkmale des Laufwerks: `VolumeName` (Name), `ShareName` (Name, unter dem das Laufwerk in einem Netzwerk angesprochen wird), `TotalSize` und `FreeSpace` (gesamte und freie Kapazität), `FileSystem` (der Dateisystemtyp als Zeichenkette, etwa ”FAT”, ”NTFS” oder ”CDFS”) und `DriveType (Fixed, Remote, Removeable` etc.)

- Die `Drives`-Aufzählung enthält nur lokale Laufwerke (bzw. mit Laufwerksbuchstbaben eingebundene Netzwerklaufwerke). Eventuell zugängliche Netzwerkverzeichnisse werden dagegen nicht erfasst!
- Das Beispiel zeigt die wichtigsten Informationen zu allen verfügbaren Laufwerke an. Wenn sich in A: keine Diskette befindet, wird dieses Laufwerk dank `On Error` übersprungen.

'Liste aller Laufwerke + verfügbaren Speicher anzeigen

```
Public fso As New FileSystemObject

Private Sub btnShowDrives_Click()
  Dim dr As Drive
  Dim rng As Range
  Dim i&
  Set rng = Me.[A1]
  rng.CurrentRegion.Clear
  On Error Resume Next
  i = 1
  For Each dr In fso.Drives
    rng.Cells(i, 1) = dr
    rng.Cells(i, 2) = FormatNumber(dr.AvailableSpace / 1024 ^ 2, 1)
& ” MB frei”
    rng.Cells(i, 3) = ” [” & dr.VolumeName & ”, ” & dr.FileSystem &
”] ”
    i = i + 1
  Next
End Sub
```

Das aktuelle Verzeichnis

`fso.GetFolder(”.”).Path` liefert den Pfad des aktuellen Verzeichnisses

- Zur Veränderung des aktuellen Laufwerks und Verzeichnisses müssen Sie allerdings weiterhin auf einige herkömmliche Kommandos zurückgreifen: `ChDrive` wechselt das aktu-

elle Laufwerk, `ChDir` wechselt das aktuelle Verzeichnis, und `CurDir` ermittelt das aktuelle Verzeichnis (samt Laufwerk).

- Zum Wechsel des aktuellen Verzeichnisses reicht `ChDir` normalerweise nicht aus – es muss auch das Laufwerk gewechselt werden. Daher lautet die übliche Kommandoabfolge:

```
Pfad = "d:\backup"
ChDrive pfad
ChDrive pfad
```

- Wenn `pfad` allerdings auf ein Netzwerkverzeichnis zeigt (\\server\share\), gibt es Probleme. `ChDrive` kommt mit Netzwerkverzeichnissen nicht zurecht und löst einen Fehler aus. (Der kann mit `On Error Resume Next` leicht übergangen werden). `ChDir` verändert zwar das aktuelle Verzeichnis, aber nur dann, wenn das Netzwerkverzeichnis als aktuelles Laufwerk gilt (etwa beim Start eines kompilierten Visual-Basic-Programms, das auf einem Netzwerk-Server liegt). Wenn das nicht der Fall ist, gibt es unter Visual Basic keine Möglichkeit, ein Netzwerkverzeichnis zum aktuellen Verzeichnis zu machen!
- Neben dem aktuellen Verzeichnis gibt es eine ganze Reihe Excel-spezifischer Verzeichnisse, deren Pfad über diverse Eigenschaften des Excel-`Application`-Objekts ermittelt werden kann.

Temporäres Verzeichnis

```
fso.GetSpecialFolder(TemporaryFolder)
```

Windows-Verzeichnis:

```
fso.GetSpecialFolder(WindowsFolder)
```

Systemverzeichnis:

```
fso.GetSpecialFolder(SystemFolder)
```

- Wenn Sie nicht nur den Namen des temporären Verzeichnisses brauchen, sondern auch einen Vorschlag für einen gültigen (noch nicht verwendeten) Dateinamen darin, verwenden Sie einfach `fso.GetTempName()`. Diese Methode liefert allerdings nur den Namen, das dazugehörige Verzeichnis müssen Sie immer noch durch `GetSpecialFolder` ermitteln.

Eigenschaften von Verzeichnissen (Folder-Objekt)

Zugriff auf Verzeichnis:

```
Dim f As Folder
Set f = fso.GetFolder("C:\Windows\System32")
```

Jetzt kann auf eine Menge Eigenschaften zugegriffen warden:
- `Name`
- `Path` (inkl. Laufwerksangabe)
- `ShortName` bzw. ShortPath (Windows 3.1-Programme)
- `DataCreated, –LastAccessed` und `–LastModified` (erzeugt, zuletzt genutzt bzw. verändert).
- `Attributes` (Compressed, Hidden, ReadOnly)
- `Type` (Beschreibung des Verzeichnistyps, be `File`-Objekten Dateikennungen)
- `Drive` (verweist auf Laufwerk-Objekt)

- `IsRootFolder` (ob Wurzelverzeichnis, etwa bei C:\. Nur wenn das nicht der Fall ist, kann mit `ParentFolder` das übergeordnete Verzeichnis ermittelt werden. `SubFolder` verweist auf eine *`Folders`*-Aufzählung mit allen untergeordneten Verzeichnissen (sofern es welche gibt; andernfalls ist `SubFolders.Count = 0`). Die Verzeichnisnamen in `Folders`-Aufzählung sind nicht sortiert.
- `Files` verweist auf alle Dateien innerhalb des Verzeichnisses. Im Gegensatz zum herkömmlichen `Dir`-Kommando werden damit weder Unterverzeichnisse noch die Pseudodateien "." Und "…" erfasst.
- `Size` ermittelt den Platzbedarf des Verzeichnisses und berücksichtigt dabei rekursiv alle Unteverzeichnisse. Aus diesem Grund kann die Ermittlung dieser Eigenschaft einige Zeit dauern. Greifen Sei nicht unnötig auf diese Eigenschaft zu.
Der resultierende Wert enthält die Summe der Byteanzahl aller Dateien. Tatsächlich ist der Platzbedarf auf der Festplatte meist grösser, weil Dateien immer sektorenweise gespeichert werden. (Eine 3 Byte lange Datei beansprucht daher je nach Dateisystem ein oder mehrere kByte Festplattenkapazität.) Der tatsächlich Platzbedarf kann allerdings auch kleiner sein, nämlich dann, wenn die Dateien (etwa in einem NT-Dateisystem) komprimiert sind. Betrachten Sie das Ergebnis von `Size` also mit einer gewissen Vorsicht!

- Die meisten Eigenschaften sind `read-only`, können nur gelesen, aber nicht verändert werden. Die einzigen Ausnahmen sind `Attributes` und `Name`.

Eigenschaften von Dateien (File-Objekt)

- Im Gegensatz zur dafür früher eingesetzten Funktion `Dir` besteht keine Möglichkeit, nur Dateien eines bestimmten Typs (z.B. `*.txt`) oder mit bestimmten Attributen zu suchen – das müssen Sie innerhalb der Schleife selbst tun.

- `Files.Count` liefert die Anzahl der Dateien, die aber nur mit einer For-Each-Schleife abgearbeitet werden könnnen.

`[Short]Name, [Short]Path, Drive, ParentFolder, Attributes, DataXxx, Size, Type`
Eigenschaft "`Type`" z.B. "`*.txt`"

Dateien und Verzeichnisse erzeugen, verschieben, kopieren oder löschen

- Mit fse.CreateFolder erzeugen Sie ein neues Verzeichnis. Erwartet Zeichenkette mit vollständigem Pfad als Parameter.
- Die Methoden `Copy`, `Move` und `Delete` können gleichermassen auf `Folder` und `File`-Objekte angewandt werden. Alternativ können Sie auch `fso.CopyFile/-Folder`, `fso.DeleteFile/-Folder` sowie `fso.MoveFile/-Folder` benutzen.
- Bei den `Copy`-Operationen durch optionalen Parameter `Overwrite` angeben, ob vorhandene Dateien und Verzeichnisse überschrieben werden sollen. Vorsicht, die Defaulteinstellung ist `True`.
- Bei der `Move`-Methode wird die Operation nur durchgeführt, wenn die Zieldatei nicht existiert. (Dieses Sicherheitsmerkmal kann nicht durch optionale Parameter beeinflusst werden)

Verzeichnisbaum rekursiv abarbeiten

- Oft wollen sie nicht nur alle Dateien innerhalb eines Verzeichnisses bearbeiten (durchsuchen, kopieren etc.), sondern auch alle Dateien in Unterverzeichnissen. Im Regelfall ist es

dazu sinnvoll, ein rekursives Unterprogramm zu formulieren, das zuerst all Dateien im aktuellen Verzeichnis bearbeitet und sich dann selbst mit den Pfaden aller Unterverzeichnisse aufruft.

```
Sub processFile(fld As Folder)
  Dim subfld As Folder, fil As File
  For Each fil In fld.Files
    'Dateien bearbeiten
  Next
  For Each subfld In fld.SubFolders
    processFile subfld
    'rekursiver Aufruf für alle Unterverzeichnisse
  Next
End Sub
```

Hilfsfunktionen

- Über das `fso`-Objekt können diverse Methoden aufgerufen werden, die bei der Analyse bzw. Synthese von Dateinamen hilfreich sind. Alle hier beschriebenen Methoden erwarten Zeichenketten als Parameter und liefern eine Zeichenkette als Ereignis (also keine `File`- oder `Folder`-Objekte).

BuildPath(pfad, name)	bildet aus Pfad und Name einen vollständigen Dateinamen
GetAbsolutePath(name)	liefert den vollständigen Dateinamen, wenn nur ein Name rekursiv zum aktuellen Verzeichnis gegeben ist
GetBaseName(name)	liefert den einfachen Dateinamen (ohne Verzeichnis / Laufwerk)
GetDriveName(name)	liefert das Laufwerk
GetFileName(name)	wie *GetBaseName*
GetParentFolderName(name)	liefert das Verzeichnis (inkl. Laufwerk, aber ohne den Dateinamen)
DriveExist(name)	testet, ob das Laufwerk existiert (True / False)
FileExist(name)	testet, ob Datei existiert
FolderExist(name)	testet, ob Verzeichnis existiert

10.9 Arbeitsmappen und Dokumenteigenschaften

Arbeitsmappen-Bearbeiter ermitteln

```
Dim s As String
Dim DatName As String
 s = ActiveWorkbook.Author
 DatName = ActiveWorkbook.Name
 MsgBox "Der Anwender " & s & " hat die Datei " & DatName & _
   " angelegt!"
```

Zugriffsdaten einer Arbeitsmappe ermitteln

```
Dim fsO As Object
Dim sName As Object
Dim sMeldung As String
    Set fsO = CreateObject("Scripting.FileSystemObject")
    Set sName = fsO.GetFile(ActiveWorkbook.Name)
```

```
MsgBox (”Dateiname: ” & ActiveWorkbook.Name & Chr(13) & _
  ”Autor: ” & ActiveWorkbook.Author & Chr(13) & _
  ”angelegt am: ” & sName.DateCreated & Chr(13) & _
  ”Letzter Zugriff: ” & sName.DateLastAccessed & Chr(13) & _
  ”Letzte Änderung: ” & sName.DateLastModified), _
  vbInformation, ”Datei-Info”
```

- Um Dateiinformationen einer Arbeitsmappe abzurufen, erstellen Sie ein FileSystemObject mit der Funktion CreateObject. Damit haben Sie Zugriff auf die Microsoft Scripting Runtime-Bibliothek. Diese Bibliothek beinhaltet einige Eigenschaften, die Sie abfragen können.
- Über Extras / Verweise einen Verweis auf die Microsoft Scripting Runtime-Bibliothek machen. Dann im Objektkatalog Scripting auswählen und alle Objekte mit deren Methoden und Eigenschaten anschauen.

- `neuDatei = Application.GetOpenFilename(”Textdateien, *.txt”)`
- `Workbooks.OpenText Filename:= neuDatei`
- `Workbooks.Open Filename:= neuDatei`

10.10Syntaxzusammenfassung zum File System Objects

FileSystemObject – Eigenschaften

Drives	verweis auf Aufzählung aller Laufwerke

FileSystemObject - Methode

BuildPath(pfad, name)	bildet vollstängigen Dateinamen
CopyFile/-Folder	Datei oder Verzeichnis kopieren
DeleteFile/-Folder	Datei oder Verzeichnis löschen
DriveExist(name)	testet, ob Laufwerk existiert
FileExist(name)	testet, ob Datei existiert
FolderExist(name)	testet, ob Verzeichnis existiert
GetAbsolutePath(relname)	bildet vollständigen Dateinamen (aus relativer Angabe)
GetBaseName(name)	liefert einfachen Namen (ohne Verzeichnis / Laufwerk)
GetDrive	liefert *Drive*-Objekt
GetDriveName(name)	liefert Laufwerksnamen
GetFile	liefert *File*-Objekt
GetFileName(name)	wie *GetBaseName*
GetFolder	liefert *Folder*-Objekt
GetParentFolderName(name)	liefert Verzeichnisnamen (mit Laufwerk)
GetSpecialFolder	liefert *Folder*-Objekt für Windows-(System-)Verzeichnis
GetTempName	liefert Namen für eine temporäre Datei (ohne Verzeichnis)
MoveFile/-Folder	Datei / Verzeichnis verschieben / umbenennen
OpenTextFile	öffnet eine Textdatei

Drive – Eigenschaft

AvailableSpace	freie Speicherkapazität
DriveType	Laufwerktyp (z.B. *Remote, CDRom* etc.)

FileSystem	Dateisystem (z.B. *"NTFS", "FAT"* etc.)
FreeSpace	wie *AdvailableSpave*
IsReady	bereit (bei A: Diskette eingelegt)
Path	Zeichenkette des Pfads ohne \ (z.B. "C:")
RootFolder	Verweis auf *Folder*-Objekt
ShareName	Laufwerksnamen im Netzwerk
TotalSize	Gesamtkapazität
VolumeName	Laufwerksnamen

File / Folder – Gemeinsame Eigenschaften

Attributes	Attribute (schreibgeschützt, komprimiert etc.)
DataCreated	Datum und Zeit der Erzeugung
DataLastAccessed	Datum und Zeit des letzten Zugriffs
DataLastModified	Datum und Zeit der letzten Änderung
Drive	Verweis auf Laufwerk (*Drives*-Objekt)
Files	Aufzählung aller enthaltenen Dateien (nur *Folder*)
IsRootFolder	*True*, wenn Wurzelverzeichnis (nur *Folder*)
Name	Name (ohne Verzeichnis / Laufwerk)
ParentFolder	Verweis auf übergeordnetes Verzeichnis (*Folder*-Objekt)
Path	Zeichenkette mit vollständigem Namen (inkl. Verz. / Laufw.)
ShortName	Name in 8+3-Konvention (DOS-Windows 3.1)
ShortPath	Pfad in 8+3-Konvention (DOS-Windows 3.1)
Size	Dateigrösse bzw. Summe der enthaltenen Dateien
SubFolders	Aufzählung aller Unterverzeichnisse (nur *Folder*)
Type	Bezeichnung des Dateyps

File / Folder – Gemeinsame Methoden

Copy	Datei / Verzeichnis kopieren
CreateTextFile	Textdatei erzeugen (nur *Folder*)
Delete	Datei / Verzeichnis löschen
Move	Datei / Verzeichnis umbenennen bzw. verschieben
OpenAsStream	als Textdatei öffnen (nur *File*)

TextStream – Eigenschaft

AtEndOfLine	Zeilenende erreicht?
AtEndOfStream	Dateiende erreicht?
Column	aktuelle Position innerhalb der Zeile
Line	aktuelle Zeilennummer

TextStream – Methode

Close	Datei schliessen
Read	n Zeichen lesen
ReadAll	die gesamte Datei in eine Zeichenkette lesen
ReadLine	die nächste Zeile lesen
Skip	n Zeichen überspringen
SkipLine	Zeile überspringen
Write	Zeichenkette schreiben (ohne Zeilenumbruchzeichen)
WriteLine	eine Zeile schreiben (mit Zeilenumbruchzeichen)
WriteBlankLines	n leere Zeilen schreiben

11 Ereignisse

11.1 Diverses

Diverses

- In Excel sind rund 50 Ereignisse definiert.
- Im Objektkatalog durch einen gelben Blitz gekennzeichnet
- Durch Listenfeld automatisch Codeschablone einfügen
- `Auto_Open` und `Auto_Close` stehen nur noch aus Kompatibilitätsgründen zur Verfügung. RunAutoMacros
- `OnAction` für das Shape-Objekt und diverse `CommandBar`-Objekte
- Öffnen-Ereignis unterdrücken mit Shift-Taste

Ereignisprozedur deaktivieren

- Aufruf einer Ereignisprozedur vorübergehend deaktivieren:
- Durch `activateEvents` (`Events` = Ereignisse) kann gesteuert werden, ob Ereignisse verarbeitet werden sollen (True) oder nicht. Sie müssen zur Veränderung von activateEvents den internen Namen des Tabellenblatts voranstellen, also etwa
- `Tabelle1.actiavteEvents = True`

```
Public activateEvents
Private Sub Worksheets_Activate()
  If activateEvents <> True Then Exit Sub
  …
End Sub
```

Alle Ereignisprozeduren deaktivieren

- Wenn Sie alle Ereignisprozeduren vorübergehend deaktivieren möchten (und nicht nur eine einzelne Prozedur wie im Beispiel oben), können Sie die Eigenschaft ***EnableEvents*** für das Applications-Objekt auf *False* setzen.
- `Application.EnableEvents = False`

Rückgängig und Wiederholen(Methoden OnUndo, OnRepeat)

- Durch OnUndo und OnRepeat Prozeduren angeben, die Excel ausführt, wenn der Anwender die Kommandos ”Bearbeiten / Rückgängig oder Wiederholen” ausführt.

- `Application.OnRepeat ”Wiederholung: Daten analysieren”, ”Makroxxx”`

- Der erste Parameter gibt den im Menü angezeigten Text an
- Der zweite Parameter gibt die Prozedur an, die ausgeführt werden soll.

Ereignisse beliebiger Objekte empfangen

- Im Klassenmodul eine öffentliche Variable definieren, deren Ereignisse Sie empfangen möchten. Dabei verwenden Sie das Schlüsselwort `WithEvents`. In den Listenfeldern des Modulfensters können Sie daraufhin alle für dieses Objekt bekannten Ereignisse auswählen.

```
' Klassenmodul ”ereignisklasse”
Public WithEvents x As Objname
Private Sub x_Ereignisname(parameterliste)
  ' … die Ereignisprozedur
```

```
End Sub
```

- Damit nun tatsächlich Ereignisse empfangen werden, muss zuerst ein Objekt der neuen Klasse und darin wiederum ein Objekt der Klasse mit den Ereignissen erstellt werden.

```
' in einem beliebigen Modul
Dim Obj As New Ereignissklasse  'obj ist ein Objekt der "ereignisklasse"
Sub startevents()
  Set obj.x = [New] objename     'x ist ein Objekt der Klasse "objname"
End Sub
```

- Nachdem `startevents` ausgeführt wurde, wird die Ereignisprozedur im Klassenmodul solange ausgeführt, bis `obj.x` oder überhaupt `obj` gelöscht wird (also `Set obj = Nothing`)

... Beispiel …
`StopApplicationEvents` beendet die automatische Veränderung neuer Arbeitsmappen

```
Option Explicit
Dim appObject As New ClassAppEvents

' startet die Ereignisprozeduren
Sub InitializeApplicationEvents()
  Set appObject.app = Application
End Sub

'beendet die Ereignisprozeduren wieder
Sub StopApplicationEvents()
  Set appObject.app = Nothing
End Sub
```

Ereignisprozeduren per Programmcode erzeugen

- Sie können neue Klassen mit eigenen Ereignissen programmieren. Das hier beschriebene Einfügen von Ereignisprozeduren in Excel-Dateien hat damit nichts zu tun.

- `ThisWorkbook.VBProject` ist eine Kurzschreibung für `Application.VBE.VBProject(ThisWorkbook.Name)`

Virengefahr durch Autoprozeduren

- Ereignisprozeduren und die `Auto_Open`- bzw. `Auto_Close`-Prozeduren sind der Schlüssel zur Viren-Programmierung in VBA
- Dank `DLL`-Unterstützung kann man fast alle Betriebssystemfunktionen aufrufen.
- Dank `Active-X`-Automation kann jede am Rechner installierte Objektbibliothek genutzt werden.

- Um die Benutzeroberfläche von Excel zu erweitern, gibt's zwei Add-In-Typen:
  - Add-Ins (*.xla)
  - COM-Add-Ins(*.dll)

Makro starten am Monatsende

- Damit Sie den Monatsabschluss nicht vergessen, können Sie Excel dazu bringen, Sie an diesen Termin zu erinnern.
- Um diese Aufgabe zu realisieren, müssen Sie zuerst einmal den letzten Tag des jeweiligen Monats ermitteln. Dazu setzen Sie eine Funktion ein, die wie folgt aussieht:

- Funktion

```
Function LetzterTag(EingabeDatum As Date) As Date
  LetzterTag = DateSerial(Year(EingabeDatum), Month(EingabeDatum) +
1, 0)
End Function
```

- Da Sie die Funktion mehrmals verwenden möchten, übergeben Sie der Funktion als Argument jeweils das aktuelle Datum über die Funktion `Date`. Die Funktion `Date` übergeben Sie gleich nach dem Öffnen der Arbeitsmappe. Dazu schreiben Sie das Ereignis `Workbook_Open`.

- Ereignis

```
Private Sub Workbook_Open()
 If LetzterTag(Date) = Date Then Abschluss
End Sub
```

- Entspricht das aktuelle Datum dem Datum, welches die Funktion ”LetzterTag” ermittelt, wird das Makro ”Abschluss” gestartet. Dieses Makro erfassen Sie auf Modulebene.

- Makro

```
Sub Abschluss()
 MsgBox ”Den Monatsabschluß starten!”
End Sub
```

Arbeitsmappe bedingt schliessen

Nur wenn in Zelle `A1` ein Wert ungleich `1` steht, kann die Arbeitsmappe geschlossen werden

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    If Range(”A1”).Value = 1 Then
        Cancel = True
    End If
End Sub
```

Letztes Bearbeitungsdatum festhalten

- Diese Information kann entweder über die Eigenschaft `BuiltinDocumentProperties` ermittelt werden oder auch über das Ereignis `Workbook_BoforeSave`. Genau vor dem Eintritt dieses Ereignisses können Sie das aktuelle Datum in eine bestimmte Zelle schreiben.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
Cancel As Boolean)
  ThisWorkbook.Sheets(1).Range(”A1”).Value = Now()
End Sub
```

Schnell Tabellen gleicher Struktur vergleichen

- Wenn Sie auf eine andere Tabelle wechseln, stellt das Makro die Markierung gleich ein wie bei der vorherigen Tabelle.

- Globale Variable

```
Dim BlattAlt As Object
```

- Workbook_SheetActivate-Ereignis

```
Private Sub Workbook_SheetActivate(ByVal Blatt As Object)
Dim lCol As Long
Dim lRow As Long
Dim sCell As String
Dim sSelection As String

  On Error GoTo Ende
  If TypeName(Blatt) <> "Worksheet" Then Exit Sub
  Application.ScreenUpdating = False
  Application.EnableEvents = False

  BlattAlt.Activate
  lCol = ActiveWindow.ScrollColumn
  lRow = ActiveWindow.ScrollRow
  sSelection = Selection.Address
  sCell = ActiveCell.Address

  Blatt.Activate
  ActiveWindow.ScrollColumn = lCol
  ActiveWindow.ScrollRow = lRow
  Range(sSelection).Select
  Range(sCell).Activate
Ende:
  Application.EnableEvents = True
End Sub
```

- Workbook_SheetDeactivate-Ereignis

```
Private Sub Workbook_SheetDeactivate(ByVal Blatt As Object)
  If TypeName(Blatt) = "Worksheet" Then Set BlattAlt = Blatt
End Sub
```

- Sie benötigen eine globale Variable, in der Sie jeweils den alten Zustand der Markierung speichern. Speichern Sie die Positionsdaten des Mauszeigers, indem Sie die Eigenschaften `ScorllColumn` und `ScrollRow` verwenden. Die Eigenschaft `ScrollColumn` gibt die Nummer der Spalte, die sich auf der linken Seite des Ausschnitts oder des Fensters befindet, zurück, die in der Variablen "sCol" gespeichert wird. Mit der Eigenschaft `ScrollRow` ermitteln Sie die Nummer der Zeile, die sich auf der oberen Seite des Ausschnitts oder des Fenster befindet, und speichern sie in der Variable "lRow". Die Adresse der aktiven Zelle bzw. der markierten Zelle speichern Sie in der String-Variablen "sSelection" und "sCell" mit Hilfe der Eigenschaft `Adress`. Im Anschluss an das Auslesen der Position des "alten" Tabellenblatts wird wieder auf das neue Tabellenblatt gewechselt.Nun werden die Posititionen aus den Variablen auf das neue Tabellenblatt übertragen.

Im Ereignis `Workbook_SheetDeaktivate` sorgen Sie dafür, dass die globale Variable *BlattAlt* mit dem Namen des neuen Tabellenblatts gefüllt wird

Die Lösung für das sparsame Drucken

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
Dim i As Integer
  i = MsgBox (”Haben Sie die Seitenansicht kontrolliert?”, vbYesNo,
”Drucken”)
  If i <> 6 Then Cancel = True: Exit Sub
End Sub
```

Druckstatistick führen

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
  With Worksheets(1).Cells(1, 1)
    .Value = .Value + 1
  End With
End Sub
```

Ereignisse der Tabellenblätter werden immer vor den Ereignissen der Arbeitsmappe ausgefürt. Auch innerhalb der Tabellenereignisse gibt es eine bestimmt Hierarchie der Ereignisse

Vergelich von zwei Spalten

Der jeweils kleinere Wert soll mit der Hintergrundfarbe Rot belegt werden.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
If Target.Column = 1 Then
    If Target.Value < Target.Offset(0, 1).Value _
    Then
      Target.Interior.ColorIndex = 3
      Target.Offset(0, 1).Interior.ColorIndex = 2
    Else
      Target.Interior.ColorIndex = 2
      Target.Offset(0, 1).Interior.ColorIndex = 3
    End If
End If
If Target.Column = 2 Then
    If Target.Value < Target.Offset(0, -1).Value _
    Then
    Target.Interior.ColorIndex = 3
    Target.Offset(0, -1).Interior.ColorIndex = 2
    Else
    Target.Interior.ColorIndex = 2
    Target.Offset(0, -1).Interior.ColorIndex = 3
    End If
End If
End Sub
```

Änderungen in einer Tabelle sichtbar machen

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
  If Intersect(Target, Range(”B2:C13”)) Is Nothing _
  Then Exit Sub
  Target.Interior.ColorIndex = 15
End Sub
```

- Über die Methode `Intersect` prüfen Sie, ob die geänderte Zelle im definierten Zielbereich `B2:C13` liegt. Wenn ja, dann färben Sie die Zelle mit der Eigenschaft `ColorIndex`, der Sie den Wert `15` (Grau) zuweisen.

- Es empfiehlt sich, diese Formatierung der geänderten Zellen vor dem Speichern der Arbeitsmappe wieder zu normalisieren:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As
Boolean)
  Sheets(”Kosten”).Range(”B2:C13”). _
  Interior.ColorIndex = xlNone
End Sub
```

Unterschiedliche Makros je nach Zellenwert starten

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
  If Target.Address = ”$A$1” And Target.Value = ”EU” Then
   Call Makro1
 Else If Target.Address = ”$A$1” And Target.Value = ”DM” Then
   Call Makro2
End Sub
```

Automatisch aktuelles Datum in zweite Spalte schreiben

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim l As Long
 l = ActiveSheet.UsedRange.Rows.Count
    If Not Application.Intersect(Target, Range(”A4:A” & l)) Is Noth-
ing Then
        Target.Offset(0, 1).Value = Now
    End If
End Sub
```

- Der Geltungsbereich soll in der ganzen Spalte A gelten, mit Ausnahme der ersten drei Zellen dieser Spalte. Ermitteln Sie aus diesem Grund zuerst einmal die Anzahl der belegten Zellen in Spalte A. Prüfen Sie im nächsten Schritt über die Methode `Intersect`, ob die Zelleneingabe im Zielbereich liegt. Wenn ja, fügen Sie in der Nebenzelle das aktuelle Datum und die Zeitüber die Funktion *Now* ein.

Wenn in Spalte A links in der Zelle ”Alt” steht, Zeile automatisch ausblenden

```
Private Sub Worksheet_Change(ByVal Target As Range)
  If Target.Column = 1 And Left(Target.Value, 3) = ”Alt” _
  Then Target.EntireRow.Hidden = True
End Sub
```

Zellen nach der Eingabe schützen

```
Private Sub Worksheet_Change(ByVal Target As Range)
  If Target.Value <> ”” Then
    ActiveSheet.Unprotect
    Target.Locked = True
    ActiveSheet.Protect
  End If
End Sub
```

Tabellennamen aus Zelle herleiten

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
Dim s As String
  If Target.Address = ”$A$1” And IsDate(Target.Value) Then
  s = Range(”A1”).Value
  s = Format(s, ”mmm yy”)
  ActiveSheet.Name = s
 End If
End Sub
```

Schrifart vergrössern bei aktiver Zelle

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
Static Zelle As Range
  If Not Zelle Is Nothing Then
    Zelle.Font.Size = 10
  End If
  Target.Font.Size = 12
  Set Zelle = Target
End Sub
```

- Setzen sie die Anwesung `Static` ein, um eine Variable zu deklarieren, die jeweils die Adresse der aktiven Zelle aufnehmen soll. Die mit Static-Anweisung deklarierte Variable behält ihren Wert bei, solange der Code ausgeführt wird. Mit dem Schlüsselwort `Nothing` überprüfen Sie, ob die Zelle verlassen wurde. Wenn ja, muss der Schriftgrad `10` gesetzt werden. Wird eine neue Zelle aktiviert, wird der Schriftgrad `12` eingestellt. Am Ende setzen Sie die Objektvariable ”Zelle” mit der aktiven Zelle gleich.

Markierung einer Zelle auf die ganze Zeile ausweiten

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
  Application.EnableEvents = False
  Rows(Target.Row).Select
  Target.Activate
  Application.EnableEvents = True
End Sub
```

- Markieren Sie die ganze Zeile. Danach muss der Mauszeiger wieder die ursprünglich markiete Zelle markieren.

Zellenkontextmenü auf einem bestimmten bereich unterdrücken

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel
As Boolean)
Dim VerbotenerBereich As Range
  Set VerbotenerBereich = Range(”B2:D10”)
  If Intersect(Target, VerbotenerBereich) Is Nothing Then Exit Sub
  Cancel = True
End Sub
```

Wenn auf Hyperlink geklickt wird, aktuelle Arbeitsmappe schliessen

- Die entsprechende Seite wird geöffnent, zudem wird gleich noch die aktuelle Arbeitsmappe geschlossen.

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
```

```
 Application.DisplayAlerts = False
 Workbooks("TEreign15.xls").Close
  Application.DisplayAlerts = True
End Sub
```

Doppelklicks deaktivieren

- Damit man nicht in den Eingabemodus der Zelle gelangt.

```
Private Sub Worksheet_Activate()
 Application.OnDoubleClick = "KeineAktion"
End Sub

Sub KeineAktion()
End Sub

Private Sub Worksheet_Deactivate()
  Application.OnDoubleClick = ""
End Sub
```

- Weisen Sie der Eigenschaft `OnDoubleClick` einen Leerstring zu, um die Funktionalität des Doppelklicks wiederherzustellen.

11.2 Reaktion auf Tastendruck mit der OnKey-Methode

- Führt die angegebene Prozedur aus, wenn eine bestimmte Taste oder Tastenkombination gedrückt wird.
- Das Argument `Key` kann eine einzelne Taste angeben, kombiniert mit den Tasten ALT, STRG oder UMSCHALT oder jede Kombination dieser Tasten.

Syntax der OnKey-Methode

```
Application.OnKey(Key, Prezedur)
```

Tabelle mit Tasten und deren Codebelegung

RÜCKTASTE	`{BACKSPACE}` oder `{BS}`
UNTBR	`{BREAK}`
FESTSTELLTASTE	`{CAPSLOCK}`
ENTF	`{CLEAR}`
ENTFERNEN oder ENTF	`{DELETE}` oder `{DEL}`
NACH-UNTEN-TASTE	`{DOWN}`
ENDE	`{END}`
EINGABETASTE (Zehnertastatur)	`{ENTER}`
EINGABETASTE	~ (Tilde)
ESC	`{ESCAPE}` oder `{ESC}`
HILFE	`{HELP}`
POS1	`{HOME}`
EINFG	`{INSERT}`
NACH-LINKS-TASTE	`{LEFT}`
NUM	`{NUMLOCK}`
BILD-AB	`{PGDN}`
BILD-AUF	`{PGUP}`
EINGABETASTE	`{RETURN}`
NACH-RECHTS-TASTE	`{RIGHT}`

ROLLEN	{SCROLLLOCK}
TAB	{TAB}
NACH-OBEN-TASTE	{UP}
F1 bis F15	{F1} bis {F15}

- Sie können jede beliebige Tastenkombination mit UMSCHALT, STRG und ALT angeben. Kombinieren Sie eine Taste mit einer oder mehreren anderen Tasten entsprechend der folgenden Tabelle.

Kombinieren mit Vorangestelltes Zeichen

UMSCHALT	+ (Pluszeichen)
STRG	^ (Caret)
ALT	% (Prozentzeichen)

- Um einem der Sonderzeichen (+,^ ,% usw.) eine Prozedur zuzuweisen, schließen Sie das Zeichen in geschweifte Klammern ein. Genauere Informationen finden Sie im zugehörigen Beispiel.

- Beispiele

- In diesem Beispiel wird der Tastenfolge STRG+PLUSZEICHEN ”Makro1” und der Tastenfolge UMSCHALT+STRG+NACH-RECHTS ”Makro2” zugewiesen.

```
Application.OnKey "^{+}", "Makro1"
Application.OnKey "+^{RIGHT}", "Makro2"
```

- In diesem Beispiel wird UMSCHALT+STRG+NACH-RECHTS **auf die normale Bedeutung zurückgesetzt**.

```
Application.OnKey "+^{RIGHT}"
```

- In diesem Beispiel wird die Tastenfolge UMSCHALT+STRG+NACH-RECHTS **deaktiviert**.

```
Application.OnKey "+^{RIGHT}", ""
```

Tastenkombinationen ein- und ausschalten

- Auf dem Tabllenblatt soll nicht mit Tastenkombinationen gearbeitet werden können.

```
Sub TastenkombinationenAusschalten()
Dim i As Integer
  On Error Resume Next
  For i = 1 To 255
    Application.OnKey "^" & Chr(i), ""
  Next i
End Sub
```

Esc-Taste deaktivieren

```
Sub ESCTasteDeaktivieren()
 Application.EnableCancelKey = xlDisabled
End Sub
```

Esc-Taste aktivieren

```
Sub ESCTasteAktivieren()
 Application.EnableCancelKey = xlInterrupt
End Sub
```

11.3 Tastenbefehle simulieren mit der Methode SendKeys

Es gelten die selben Codebelegungen wie bei der `OnKey`-Methode

Zahlenwerte nach Textimport richtig erkennen mit SendKeys

- In manchen Fällen erkennt Excel nach einem Import von Textdateien keine Zahlenwerte mehr. Dann werden Zahlen von Excel als Text interpretiert.
- Entweder in einer leeren Zelle eine 1 einschreiben, die Zelle kopieren, dann den Zahlenbereich markieren, Kontextmenü: ”Inhalte einfügen” und die Option: ”Multiplizieren” wählen. Oder folgendes Makro anwenden:

```
Sub F2ImBereich()
Dim Zelle As Object
  For Each Zelle In Selection
    SendKeys ”{F2}”, True
    SendKeys ”{ENTER}”, True
  Next Zelle
End Sub
```

Siehe auch Tastatureingaben in Dialogen simulieren mit SendKeys

auch auf Seite 58

11.4 Zeitsteuerung in Excel mit der OnTime-Methode

Die Syntax der Methode OnTime

OnTime(EarliestTime, Procedure, LatestTime, Schedule)

- Das Argument `EarliestTime` gibt den Zeitpunkt an, an dem eine Prozedur ausgeführt werden soll. Das Argument `Procedure` beinhaltet den Namen der auszuführenden Prozedur. Das Argument `LatestTime` ist optional einsetzbar und gibt den letzten Zeitpunkt an, zu dem die Prozedur ausgeführt werden kann. Das letzte Argument `Schedule` ist optional einsetzbar und führt eine neue OnTime-Pro

Diverses über die OnTime-Methode

- Prozedur zu einem bestimmten Zeitpunkt starten.
  `Application.OnTime Now + TimeValue(”0:00:30”), ”Makro”`
- Über OnTime dürfen mehrere Prozeduren für verschiedene Zeitpunkte vorgemerkt werden.
- Die Ausführung kann sich verzögern, wenn Excel gerade mit anderen Dingen beschäftigt ist.
- Durch den dritten Parameter kann der späteste Zeitpunkt angegeben werden, zu dem die Prozedur gestartet wird. Verstreicht dieser Zeitpunkt, ohne das Excel die Möglichkeit hat, die Prozedur aufzurufen, wird auf einen Aufruf ganz verzichtet.
- Durch die Angabe von `False` im vierten Parameter kann eine terminierte Prozedur wieder gelöscht werden. Dazu muss die anvisierte Zeit als auch der Name der Prozedur angegeben werden.

```
‘Makro für 8:30 anmelden
▪  Application.OnTime #8:30#, ”Makro”
’Makro wieder abmelden
▪  Application.OnTime #8:30#, ”Makro”, , False
```

- Wenn Sie einen periodischen Aufruf einer Prozedur erreichen möchten, müssen Sie mit OnTime innerhalb der Prozedur einen weiteren Aufruf anmelden. Die folgende Prozedur ändert, wenn sie einmal gestartet ist, alle zehn Sekunden den Text der Statuszeile und zeigt dort die aktuelle Uhrzeit an.

```
Sub Statuszeile()
  Application.OnTime Now + Timevalue(”0:00:10”), ”statuszeile”
  Application.Statusbar = Now
End Sub
```

- Es ist nicht ganz einfach, diese Prozedur wieder zu stoppen. Sie können entweder den Namen der Prozedur verändern oder die `OnTime`-Anweisung durch ”’” in einen Kommentar umwandeln. Wenn Sie möchten, dass in der Statuszeile wieder die normalen Infotexte von Excel angezeigt werden, müssen Sie im Direktbereich die folgende Anweisung ausführen:

- `Application.Statusbar = False`

Regelmässig die Uhrzeit in der Statusleiste anzeigen

```
Sub Zeitanzeige()
 Application.OnTime Now + TimeValue(”00:01:00”), ”Uhrzeit”
End Sub

Sub Uhrzeit()
    Application.DisplayStatusBar = True
    Application.StatusBar = Date & ”,” & Time
    Application.Wait (Now + TimeValue(”0:00:05”))
    Application.StatusBar = False
    Call Zeitanzeige
End Sub
```

Arbeitsmappe nach 30 Sekunden schliessen

```
Sub CountdownEinstellen()
  Application.OnTime Now + TimeValue(”00:00:30”), ”ExcelEnde”
End Sub

Sub ExcelEnde()
 Application.DisplayAlerts = False
 Application.Quit
End Sub
```

Alle zehn Sekunden nachschauen, ob Arbeitsmappe im Verzeichnis ist

```
Sub MappeDa()
Dim s As String
Const AName = ”Mappe1.xls”
  s = Dir(”c:\eigene Dateien\” & AName)
  If s = ”” Then
    Call PrüfenArbeitsmappe
  Else: Workbooks.Open s
    Beep
    Exit Sub
End Sub
```

```
Sub PrüfenArbeitsmappe()
 Application.OnTime Now + TimeValue(”00:10:00”), ”MappeDa”
End Sub
```

- Sollte der Signalton nicht ausreichen, um Sie auf die Verfügbarkeit der Arbeitsmappe aufmerksam zu machen, dann können Sie folgendes Makro einsetzen

```
Sub SignaleAusgeben()
Dim i As Integer
 For i = 1 To 10
    Beep
    Application.Wait (Now + TimeValue(”0:00:01”))
 Next i
End Sub
```

Zellen blinken lassen

```
Dim Nochmal As Date

Sub Blinken()
  Nochmal = Now + TimeValue(”00:00:01”)
  With ActiveCell.Interior
    If .ColorIndex = 2 Then .ColorIndex = 4 Else .ColorIndex = 2
  End With
  Application.OnTime Nochmal, ”Blinken”
End Sub

Sub BlinkenStop()
  Application.OnTime Nochmal, ”Blinken”, schedule:=False
  ActiveCell.Interior.ColorIndex = xlColorIndexNone
End Sub
```

- Da eine normale Variable nach dem Ende eines Makros wieder zurückgesetzt, also initialisiert wird, müssen Sie eine globale Variable definieren, in welcher Sie den Zeitpunkt für das nächste Bilken der Zelle festlegen. Dieses Intervall legen Sie mit einer Sekunde fest. Danach prüfen Sie den Zellenhintergrund der aktiven Zelle. Ist dieser weiss, dann belegen Sie ihn mit der Farbe Grün, ist die Zelle jedoch in diesem Augenblick grün, dann formatieren Sie Sie den Zellenhintergrund mit der Farbe Weiss. Danach rufen Sie am Ende des Makros das Makro gleihc noch einmal auf, indem Sie die Methode *OnTime* einsetzen und als Startpunkt für das Makro den Zeitpunkt angeben, den Sie vorher in der Variablen ”Nochmal” gespeichert haben. Klar, dass es sich hierbei um eine Endlosschleife handelt, die Sie aber durch das Makro ”BlinkenStop” beenden können.

11.5 Syntaxzusammenfassung

Ereignisempfang für beliebige Objekte

```
Public With x As objname          im Klassenmodul ”eventclass”
Private Sub x_eventname(param)
  ,… die Ereignisprozedur
End Sub

Dim Obj As New eventclass         in einem beliebigen Modul
Sub startevents
  Set obj.x = [New] objname       ab jetzt können Ereignisse empfangen werden
```

End Sub

Autoprozeduren

Sub Auto_Open()	wird beim Öffnen der Datei gestartet
Sub Auto_Close()	wird beim Schliessen der Datei gestartet
Sub Auto_Add()	wird gestartet, wenn ein Add-In in die Liste des Add-In-Managers eingetragen wird
Aub Auto_Remove()	wird beim Entfernen aus der Add-In-Liste gestartet

Application-Ereignisse

NewWorkbook	eine neue Arbeitsmappe wurde eingefügt
SheetActivate	Blattwecksel
SheetBeforeDoubleClick	Doppelklick
SheetBeforRightClick	Klick mit rechter Maustaste
SheetCalculate	Tabellenblatt wurde neu berechnet
SheetChange	Zellen des Tabellenblatts wurden verändert (Inhalt)
SheetDeactivate	Blattwecksel
SheetSelectionChange	Markierungswechsel
WindowActivate	Fensterwechsel
WindowDeactivate	Fensterwechsel
WindowResize	neue Fenstergrösse
WorkbookActivate	Wechsel der aktiven Arbeitsmappe
WorkbookAddinInstall	eine Arbeitsmappe wurde als Add-In installiert
WorkbookAddinUninstall	eine Arbeitsmappe wurde als Add-In deinstalliert
WorkbookBeforeClose	eine Arbeitsmappe soll geschlossen werden
WorkbookBeforePrint	eine Arbeitsmappe soll ausgedruckt werden
WorkbookBeforSave	eine Arbeitsmappe soll gespeichert werden
WorkbookDeactivate	Wechsel der aktiven Arbeitsmappe
WorkbookNewSheet	in die Arbeitsmappe wurde ein neues Blatt eingefügt
WorkbookOpen	die Arbeitsmappe wurde geöffnet

Workbook-Ereignisse

Activate	die Arbeitsmappe wurde aktiviert (Fensterwechsel)
AddinUninstall	die Arbeitsmappe wurde als Add-In **installiert**
BeforClose	die Arbeitsmappe soll geschlossen werden
BeforPrint	die Arbeitsmappe soll ausgedruckt werden
BeforSave	die Arbeitsmappe soll gespeichert werden
Deactivate	die Arbeitsmappe wurde aktiviert (Fensterwechsel)
NewSheet	ein neues Blatt wurde eingefügt
Open	die Arbeitsmappe wurde gerade geöffnet
SheetsActivate	Blattwechsel
SheetBeforDoubleClick	Doppelkick in einem Blatt
SheetBeforRightClick	Klick mit der rechten Maustaste in einem Blatt
SheetCalculate	der Inhalt eines Blatts wurde neu berechnet
SheetChange	Eingabe oder Veränderung einer Zelle
SheetDeactivate	Blattwechsel
SheetSelectionChange	Veränderung der Markierung
WindowActivate	Fensterwechsel
WindowDeactivate	Fensterwechsel
WindowResize	Änderung der Fenstergrösse

12 Datentransfer über die Zwischenablage

Bearbeiten / Inhalte einfügen

```
Selection.PasteSpecial _
      Paste:=xlPasteValues, _         'Einfügen:=xlWert
      Operation:=xlNone, _
      SkipBlanks:=False, _
      Transpose:=False
```

CutCopyMode:

```
Select Case Application.CutCopyMode
    Case Is = False
        MsgBox "Weder im Ausschneide- noch im Kopiermodus"
    Case Is = xlCopy
        MsgBox "Kopiermodus"
    Case Is = xlCut
        MsgBox "Ausschneidmodus"
End Select
```

ClipboardFormats

- Diese Aufzählungseigenschaft gibt an, welche Formate die in der Zwischenablage befindlichen Daten aufweisen. Die Eigenschaft ist als Feld orgnanisiert, weil die Zwischenablage gleichzeitig Daten in mehreren Formaten enthalten kann. Mögliche Formate sind `xlClipboardFormatText` oder `xlClipboardFormatBitmap`.

Sichtbare Zellen von Tabelle 1 nach Tabelle 2 kopieren

```
Selection.CurrentRegion.SpecialCells(xlVisible).Copy
ActiveSheet.Paste Range("Tabelle2!A1")
Application.CutCopyMode = False
```

Zugriff auf die Zwischenablage mit dem DataObject:

```
Dim neuDataObj As New DataObject
```

- Inhalt der Zwischenablage mit der Methode **`GetFromClipboard`** in dieses Objekt kopieren. Umgekehrt können Sie **`PutInClipboard`** verwenden, um den Inhalt von neuDataObj in die Zwischenablage zu übertragen.

```
Dim neuClipText As String
neuDataObj.GetFromClipboard
neuClipText = neuDataObj.GetText
```

- Der umgekehrte Weg, also das Kopieren eines Textes in die Zwischenablage, sieht folgendermassen aus:

```
neuDataObj.Clear
neuDataObj.PutInClipboard
```

12.1 Syntaxzusammenfassung

Zellbereiche kopieren/ausschneiden/einfügen

Range.Copy	Bereich in die Zwischenablage kopieren
Range1.Copy Range2	Daten aus Range1 in Range2 kopieren
Range.Cut	wie Kopieren, Range wird aber gelöscht
Range1.Cut Range2	Daten aus Range1 in Range2 versetzten
wsheet.Paste	fügt Daten in das Tabellenblatt ein
wsheet.Paste Link:=True	wie oben, aber mit bleibender Verknüpfung
wsheet.Paste Range	fügt Daten im angegebenen Bereich ein
wsheet.Special format	fügt Daten in bestimmtem Format ein
Application.CutCopyMode	gibt den aktuellen Modus an
Application.ClipboardFormats(n)	enthält Informationen über Daten in der Zwischenablage

MSForms.DataObject – Methode

Clear	Inhalt des Objekts löschen
GetFromClipboard	Inhalt des Objekts aus der Zwischenablage lesen
PutInClipboard	
GetFormat	ermittelt Datenformate (wie ClipboardFormats)
GetText	Text im Objekt speichern
SetText	Text aus dem Objekt lesen

13 Operatoren in VBA

Arithmetische Operatoren

`-`	negatives Vorzeichen
`+ - * /`	Grundrechenarten
`^`	Potenz (3^2 ergibt 9)
`\`	ganzzahlige Division
`Mod`	Module-Operation (Rest einer ganzzahligen Division)

Operatoren zur Verknüpfung von Zeichenketten

`+`	nur für Zeichenketten
`&`	Zahlen werden automatisch in Zeichenketten umgewandelt

Vergleichsoperatoren

`=`	gleich
`<>`	ungleich
`< <=`	kleiner bzw. kleiner-gleich
`> >=`	grösser bzw. grösser-gleich
`Is`	Verweis auf dasselbe Objekt
`Like`	Mustervergleich

Logische Operatoren

`And`	logisches Und
`Or`	logisches Oder
`Xor`	exklusives Oder (entweder a oder b, aber nicht beide)
`Imp`	Implikation (wenn a wahr ist, dann mus auch b wahr sein)
`Aqv`	Äquivalent (a und b müssen übereinstimmen)
`Not`	logische Negation

Zuweisungsoperatoren

`=`	Zuweisung an Variablen und Eigenschaften
`:=`	Zuweisung an benannte Parameter beim Prozeduraufruf

14 Funktionen

14.1 Excel-Tabellenfunktionen in VBA-Programmen nutzen

- Es gibt Funktionen, die nur in VBA verwendet werden können und Funktionen, die nur in Tabellen verwendet werden können. Wieder andere, die doppelt definiert sind und daher sowohl in VBA als auch in Tabellen verwendet werden können und durch das Voranstellen von `Application.WorksheetFunction` benutzt werden dürfen.
- Sie können in VBA fast alle Excel-Tabellenfunktionen verwenden. Manche Funktionen sind sowohl in Excel als auch in VBA definiert und können ohne weitere Schlüsselwörter verwendet werden, etwa `Sin(0.7)`. Tabellenfunkionen, die in VBA keine entsprechung finden, muss `Application.WorksheetFunction` vorangestellt werden, beispielsweise `Application.WorksheetFunction.Sum(…)` zur Verwendung der SUMME-Funktion.

- Es müssen die englischen Funktionsnamen verwendet werden (siehe Objektkatalog zum `WorksheetFunction`-Objekt). Wenn Sie nicht wissen, wie der englische Name einer deutschen Funktion lautet oder wo sich die deutschsprachige Beschreibung zu einer englischen Funktion befindet, müssen Sie ein Wörterbuch zu Hilfe nehmen oder raten – weder die Online-Hilfe noch die **Übersetzungsdatei "Vbaliste.xls"** (Verzeichnis: Office2000\Office\1031' bei der deutschen Version) geben Auskunft darüber.
- In Excel 5 / 7 wurden Tabellenfunktionen direkt mit `Application.Name()` aufgerufen, d.h. ohne die erst in Excel 97 eingeführte Eigenschaft `WorksheetFunction`, die auf das gleichnamige Objekt mit iner Liste aller Tabellenfunktionen verweist. Der grösste Fortschritt von `WorksheetFunction` besteht darin, dass jetzt mehr Tabellenfunktionen als bisher in VBA verfügbar sind.
- Die Kurzschreibweise `Application.Name()` ist aus Kompatiblitätsgründen weiterhin erlaubt, die Funktionen werden im Objektkatalog nicht angezeigt. Verwenden Sie bei neuem Code `WorksheetFunction`, um mögliche möglichen Kompatiblitätsproblemen in künftigen Excel-Versionen aus dem Weg zu gehen.

Umgang mit Zahlen und Zeichenketten

- Generell tritt bei numerischen Funktionen das Problem auf, dass diese zum Teil doppelt definiert sind – zum einein in der Programmiersprache VBA und zum andern als Tabellenfunktion in Excel. Aus diesem Grund kommt es auch vor, dass es zur Lösung einer Aufgabe mehrere Funktionen gibt, die zwar ähnlich aussehen, zumeist aber nicht ganz gleich funktionieren.

Das Jahr 2000 Problem

- `Jahreszahlen zwischen 0 und 30 gehören zum 21. Jahrhundert`
- `Jahreszahlen zwischen 30 und 99 gehören zum 20. Jahrhundert`

?Year("1.1.29")
2029
?Year("1.2.31)
1931
?Year("1.1.1")
2001

VLookup
Engl. Bezeichnung für `Sverweis`

- Die Tabellenfunktion muss immer auf das Objekt `Application`, also auf Excel, angewendet werden, als eine Methode dieses Objekts!

```
Public Sub HoleAutopreis()
Dim Autopreis As Variant
Dim Hersteller As String
Dim DrchSuche As Range
  Hersteller = InputBox("Hersteller")
  Set DurchSuche = Worksheets("PKW's").Range("Suchbereich")
  Autopreis = Application.VLookup(Hersteller, Suchbereich, 2)
  MsgBox (CStr(Autopreis))
End Sub
```

Csng

```
Zahl1 = CSng(InputBox("Zahl1"))
```

- `CSng wandelt Zeichenkette in Singelwert um`

14.2 Syntaxzusammenfassung

Runden

Clnt(v)	rundet bei 0.5
CLng(v)	rundet bei 0.5
Int(f)	rundet immer ab
Fix(f)	schneidet die Nachkommastellen ab
WorksheetFunction.Round(f, n)	rundet bei 0.5 auf die Stellenzahl n
WorksheetFunction.RoundDown(f, n)	rundet immer ab (n Nachkommastellen)
WorksheetFunction.RoundUp(f, n)	rundet immer auf (n Nachkommastellen)
WorksheetFunction.Even(f)	rundet zur betragsmässig grössten geraden Zahl
WorksheetFunction.Odd(f)	rundet zur betragsmässig grösseren ungeraden Zahl
WorksheetFunction.Ceiling(f1, f2)	rundet zum Vielfachen von f2 auf
WorksheetFunction.Floor(f1, f2)	rundet zum Vielfachen von f2 ab

Sonstige numerische Funktionen

Abs(f)	entfernt das Vorzeichen
Sgn(f)	liefert je nach Vorzeichen -1, 0, 1
Sqr(f)	Quadratwurzel
Sin(f), Cos(f), Tan(f)	trigonometrische Funktionen
Atn(f)	Umkehrfunktionen zu Tan
Log(f), Exp(f)	logarithmische Funktionen
Rnd	liefert Zufallszahl zwischen 0 und 1
Randomize	initialisiert den Zufallszahlengenerator

Zeichenketten

Left(s, n)	liefert die ersten n Zeichen
Right(s, n)	liefert die letzten n Zeichen
Mid(s, n)	liefert alle ab dem n-ten Zeichen
Mid(s, n1, n2)	liefert n2 Zeichen ab dem n1-ten Zeichen
Len(s)	ermittelt die Länge der Zeichenkette
InStr(s1, s2)	sucht s2 in s1; Ergebnis: Position oder 0
InStr(n, s1, s2)	wie oben, Suche beginnt mit n-ten Zeichen
InStr(n, s1, s2, 1)	wie oben, Gross- und Kleinschreibung egal
InStrRev(s1, s2 [,n])	wie InStr, aber Suche von hinten nach vorne
Split(s, ”x”)	zerlegt s an den Stellen des Zeichens ”x”; liefert Array
Join(array, ”x”)	setzt ein Array von Zeichenketten wieder zusammen (mit ”x” an den Anfügestellen)
Filter(array, ”x”)	liefert Array mit allen Zeichenketten, die ”x” enthalten
Replace(s, ”x”, ”y”)	ersetzt in s alle ”x” durch ”y”
UCase(s)	wandelt alle Klein- in Grossbuchstaben um
LCase(s)	wandelt alle Gross- in Kleinbuchstaben um
Ltrim	Löscht Leerzeichen am Anfang eines Strings
Rtrim	Löscht Leerzeichen am Ende eines Strings
Trim(s)	eliminiert Leerzeichen am Anfang und Ende
String(n, ”x”)	liefert eine Zeichenkette aus n mal ”x”
Space(n)	liefert n Leerzeichen

Option Comparison Text	dann gilt ”a” = ”A” und ”A” <”Ä”<”B”
StrComp(s1, s2)	-1 wenn s1<s2, 0 wenn s1=s2 bzw ein String leer ist
StrComp(s1, s2, 0)	sonst +1. Es wird überprüft, welche zuerst im Alphabet
StrComp(s1, s2, 1)	steht.
StrConv	Wandelt eine Zeichenkette um StrConv(”heinrich Hoffmann”, **vbProperCase**) ergibt Heinrich Hoffmann. Ebenso stehen die beien Parameter **vbLowerCase** und **vbUpperCase** zur Verfügung
MsgBox ”text	zeigt den Text in einem Dialog an
MsgBox(”text”, buttons)	wie oben; ermöglicht Auswahlentscheidung
InputBox(”text”)	ermöglicht die Eingabe einer Zeichenkette

Umwandlungsfunktionen

CInt(v)	liefert eine ganze Zahl
CLng(v)	wie oben, aber grösserer Zahlenbereich
CSng(v)	einfache Fliesskommazahl
CDbl(v)	doppelte Flieskommazahl
CCur(v)	Zahl im Währungsformat
CBool(v)	Wahrheitswert (True / False)
CDate(v)	Datum / Uhrzeit
CStr(v)	Zeichenkette
Val(s)	liefert den Wert der Zeichenkette
Str(v)	wandelt Zahl in Zeichenkette um
Format(v, s)	liefert Zeichenkette, wobei die Formatierung in s berücksichtigt werden
FormatNumber(v, n)	formatiert x als Betrag mit n Nachkommastellen
FormatCurrency(v, n)	formatiert x als Geldbetrag mit n Nachkommastellen
FormatPercent(v, n)	formatiert x als Prozentwert mit n Nachkommastellen
Asc(s)	liefet den ANSI-Code des ersten Zeichens
AscW(s)	liefert den Unicode des erstens Zeichens
Chr(n)	liefert das Zeichen zum ASCII-Code (0-255)

Datentyp festlegen

IsNumeric(variabele)	IstZahl
IsDate(variabele)	IstDatum oder IstUhrzeit
IsArray(variabele)	IstFeld
IsError(variabele)	IstFehlerwert
IsMissing(variabele)	Wurde optionaler Parameter übergeben
IsEmpty(variabele)	IstLeer
IsObject(variabele)	IstVerweisAufObjekt
VarType(variabele)	numerischer Wert, der den Datentyp angibt
TypeName(variabele)	Zeichenkette, die Daten-/ Objekttyp beschreibt

14.3 Rechnen mit Datum und Uhrzeit in VBA

- Der Umgang mit Datum und Uhrzeit war schon immer eine Angelegenheit, die einfacher aussieht, als sie in Wirklichkeit ist. Microsoft hat durch eine Unzahl von Funktionen wenig dazu beigetragen, das Ganze übersichtlicher zu machen.
- Datumasangabe im amerikanischen Format (Monat/Tag/Jahr) `#12.31.1999#`.

- Für die deutsche Schreibweise Daten und Zeiten in *CDate*-Funktion angeben, etwa **`CDate("31.12.1999")`** statt `#12.31.1999#` oder **`CDate("17:30")`** statt `#5:30:00 PM#`. Diese Form der Zeitangabe hat zwei Nachteile. Erstens wird der Code (minimal) langsamger ausgeführt, und zweitens ist der Code nicht portabel (weil die `CDate`-Konvertierung bei anderen Landeseinstellung – etwa in England oder in den Vereinigten Staaten – versagt).
- Weitere Alternative: `DateSerial(jahr, monat, tag)` oder `TimeSerial(stunde, minute, sekunde)`

- Die Zahl 0 entspricht intern der Zahl 1.1.1900
- Uhrzeiten werden in Nachkommaanteil gespeichert: 0,25 entspricht dem 1.1.1900 6:00
- Daten vor dem 1.1.1900 werden durch negative Zahlen dargestellt.

- Bei uns ist der **gregorianische Kalender** gültig. Jedes durch 4 teilbare Jahr hat einen Schalttag, etwa 1988, 1992, 1996. Durch 100 teilbare Jahre sind von dieser Regel ausgeschlossen, weswegen 1700, 1800, 1900 keinen Schalttag haben. Durch 400 teilbare Jahre sind wiederum von der Ausnahme ausgenommen, so dass in den Jahren 1600 und 2000 doch ein Schalttag auftritt.

Date, Time, Timer

- Anzahl der Sekunden seit Mitternacht

DateValue und TimeValue

- Nehmen als Argument eine Zeichenkette in der unter Windows eingestellten Sprache entgegen und liefern das Ergebnis im `Date`-Format von Excel. `DateValue("31. Dezember 1999")` liefert daher den 31.12.1999

DateSerial und TimeSerial

- Nehmen jeweils drei Argumente entgegen, entweder `Jahr, Monat, Tag` oder `Stunde, Minute, Sekunde`. Das Ergebnis ist wiederum ein `Date`-Wert. `DateSerial(1997, 12, 31)` liefert somit den 31.12.1997. Die Funktionen sind bei der Auswertung der Parameter ungeheuer flexibel. Soe liefert `DateSerial(1997, 13, 1)` den 1.1.1998, *DateSeria(1997, 2, 31)* den 3.3.1998, `DateSerial(1998, 0, -1)` den 29.11.1997. Entsprechend liefert `TimeSerial(4, -5, 0)` 3:55.

Hour, Minute und Secound

- Ermitteln die Bestandteile der Uhrzeit. Minute(#6:45:33#) liefert 45. Die Uhrzeit darf sowohl als `Date`-Wert als auch in Form einer Zeichenkette angegeben werden.

Year, Month und Day

- Sind die äquivalenten Funktionen für Jahr, Monat (1-12) und Tag (1-31) eines Datums. `Month("1.April 1999")` liefert 4.

WeekDay

- Funktioniert wie Day und liefert den Wochentag (1-7 für Sonntag bis Samstag). Sie können statt dessen auch die Tabellenfunktion `Application.WorksheetFunction.WeekDay` verwenden. Diese Funktion unterscheidet sich von der gleichnamigen VBA-Funktion durch einen zweiten optionalen Parameter für den Modus `m`. Für `m=2` liefert die Funktion die Werte `1` bis `7` für Montag bis Sonntag, für `m=3` die Werte `0` bis `6`. (Siehe

auch die Online-Hilfe zu `Wochentag`. Tabellenfunktionen müssen zwar auf englisch im VBA-Code verwendet werden, sind in der Online-Hilfe aber nur unter den deutschen Namen verzeichnet)

Day360

- Speziel zum Rechnen in Jahren mit 360 Tagen, die in manchen Branchen üblich sind, eignet sich `Application.WorksheetFunction.`**`Day360`**. Die Tabellenfunktion ermittelt die Anzahl von Tagen zwischen zwei Daten auf Basis von 12 Monaten zu je 30 Tagen. Wenn als optionaler dritter Parameter *False* angegeben wird, rechnet die Funktion nach der europäischen Methode, andernfalls (Defaulteinstellung) nach der amerikanischen. Schaue in der Online-Hilfe nach

- FALSCH oder nicht angegeben

- US-Methode (NASD). Ist das Ausgangsdatum der 31. eines Monats, wird dieses Datum zum 30. desselben Monats. Ist das Enddatum der 31. eines Monats und das Ausgangsdatum ein Datum vor dem 30. eines Monats, wird das Enddatum zum 1. des darauffolgenden Monats. In allen anderen Fällen wird das Enddatum zum 30. desselben Monats.

- WAHR

- Europäische Methode. Jedes Ausgangs- und Enddatum, das auf den 31. eines Monats fällt, wird zum 30. desselben Monats.

CDate

Entspricht im wesentlichen einer kombination aus `DateValue` und `TimeValue`

14.4 Rechnen mit Datum

DateAdd

- Um zu einem Datum oder zu einer Uhrzeit ein oder mehrere Zeitintervalle zu addieren. Das Intervall wird in Form einer Zeichenkette angegeben: ”yyyy” zum Addieren von Jahren, ”q” für Quartale, ”m” für Monate, ”ww” für Wochen, ”y”, ”w” oder ”d” für Tage, ”h” für Stunden, ”m” für Minuten und ”s” für Sekunden. Der zweite Parameter gibt an, wie oft das Intervall addiert werden soll. (Mit negativen Zahlen können Sie auch rückwärts rechnen. Es sind allerdings nur ganze Intervalle möglich, halbe oder viertel Stunden müssen Sie in Minuten rechen). Der dritte Parameter enthält die Ausgangszeit:

- `DateAdd(”yyyy”, 1, Now)` ’Datum und Zeit in einem Jahr
- `DateAdd(”h”, -2, Now)` ’Datum und Zeit vor zwei Stunden

- Wenn sich durch die Addition ungültige Daten ergeben (etwa der 31.4), ermittelt Visual Basic den ersten gültigen Tag vorher (30.4). Beachten Sie, dass sich `DateSerial` hier anders verhält und aus `DateSerial(1998, 4, 31)` den 1.5.1998 macht!

DateDiff

- Mit DateDiff können Sie auf einfache Weise ermitteln, wie viele Zeitintervalle sich zwischen Daten oder Zeiten befinden. Das Intervall wird wie bei `DateAdd` durch eine Zeichenkette angegeben. Die Online-Hilfe beschriebt im Detail, wie die Funktion rechnet. (Im Regelfall wird einfach auf das jeweilige Intervall rückgerechnet. Die Zeitdifferenz vom 31.1. zum 1.2. gilt deswegen als ganzer Monat, während die viel längere Zeitdifferenz von 1.1. zum 31.1. keinen Monat ergibt.)

- `DateDiff("m", Now, "1.1.1998")` 'Anzahl der Monate bis/vom 1.1.1998

- **Wie viele Tage sind es noch bis 1.1.2000?**

  AnazhlTage = DateDiff("d", Now, "1.1.2000")

DatePart

- Ermittelt die Anzahl der Perioden für einen bestimmten Zeitpunkt: Bei Jahren wird vom Jahr 0 aus gerechnet, bei Quratalen, Monaten, Wochen, Kalenderwochen ("ww") und Tagen ("y") vom 1.1. des Jahres, bei Monatstagen ("w") vom ersten Tag des Monats, bei Wochentagen ("d") vom ersten Tag der Woche (ohne optionale Parameter ist das der Sonntag) und bei Stunden von 0:00, bei Minuten und Sekunden von der letzten vollen Stunde oder Minute. `DatePart` erfüllt also in den meisten Fällen dieselbe Aufgabe wie die schon erwähte Funktionen `Year, Month, Day, Weekday` etc.

- `DatePart("m", Now)` 'Anzahl der Monate seit dem 1.1.
- `DatePart("y", Now)` 'Anzahl der Tage seit dem 1.1.
- `DatePart("d", Now)` 'Anzahl der Monatstage
- `DatePart("w", Now)` 'Anzahl der Wochentage

14.5 Datums-Tabellenfunktionen

HEUTE und JETZT

- `HEUTE` und `JETZT` entsprechen den VBA-Funktionen `Date` und `Now`
- `DATUM` und `ZEIT` entsprechen `DateSerial` und `TimeSerial` und setzen aus den drei Werten `Jahr / Monat / Tag` bzw. `Stunde / Minute / Sekunde` ein Datum bzw. eine Uhrzeit zusammen.

DATWERT und ZEITWERT

- `DATWERT` und `ZEITWERT` entsprechen `DateValue` und `TimeValue`. Sie wandeln Zeichenketten (etwa "3.April") in Daten bzw. Uhrzeiten um. Wenn kein Jahr angegeben wird, verwendet Excel automatisch das aktuelle Jahr. (Diese Besonderheit gilt nur für Tabellen-, nicht aber für die VBA-Funktionen).

JAHR, MONAT, TAG, STUNDE, MINUTE und SEKUNDE

- `JAHR, MONAT, TAG, STUNDE, MINUTE` und `SEKUNDE` entsprechen den VBA-Funktionen *Year, Month* etc. Die ebenfalls schon oben beschriebenen Tabellenfunktionen WOCHENTAG (in VBA `WeekDay`) und TAGE360 (also `Days360`) dürfen auch im VBA-Code verwendet werden, wenn `Application` vorangestellt wird.

Wie man ein Datum in die Zelle schreiben kann

`1-2-00` bzw. `1-2`

14.6 Rechnen mit Datum:

Eine Woche zu einem Datum addieren

`= d+7`

Einen Monat zu einem Datum addieren

```
= DATUM(JAHR(d) ;MONAT(d) + 1; TAG(d))          'Tabellenformel
DateSerial(Year(d) + 1, Month(d), Day(d))
```

Anzahl der Tage des aktuellen Monats

```
= DATUM(JAHR(d); MONAT(d) + 1; 1) – DATUM(JAHR(d) ;MONAT(d) ;1)
DateSerial(Year(d)+1, Month(d), Day(d))
```

Anzahl der Tage des aktuellen Jahrs

```
=DATUM(JAHR(d)+1; 1; 1) – DATUM(JAHR(d); 1; 1)
DateSerial(Year(d)+1, 1, 1) – DateSerial(Year(d), 1, 1)
```

Anzahl der Tage des laufenden Jahrs bis zu einem gegebenen Datum

```
=1 + d – DATUM(JAHR(d); 1; 1)
1 + d – DataSerial(Year(d), 1, 1)
```

Datum des Monatsletzten ermitteln

```
=DATUM(JAHR(d); MONAT(d); 0)
DateSerial(Year(d), Month(d) + 1, 0)
```

Zeitdifferenz in Jahren (Altersberechnung)

Die Formel `Year(d2)-Year(d1)` bzw. `JAHR(d2)-JAHR(d1)` ist unzureichend, wenn das Alter von Personen berechnet werden soll. So liefert `Year("1.1.1999")-Year("1.7.1950")` das Ergebnis `49`, obwohl die Person am 1.1.1999 erst 48 Jahre alt ist.

Die Lösung für dieses Problem ist etwas umständlich, einfacher geht es aber anscheinend nicht: Es wird die Jahresdifferenz um 1 korrigiert, wenn das aktuelle Datum d2 kleiner ist als das Geburtsdatum d1, wobei in d1 aber das Jahr d2 eingesetzt wird. Um nochmals auf das obige Beispiel zurückzukommen: Die unmittelbare Jahresdifferenz beträgt 49, da aber der 1.1.1999 "kleiner" als der 1.7.1999 ist, wird dieser Wert um 1 reduziert

```
= JAHR(d2)-JAHR(d1)-WENN(d2<DATUM(JAHR(d2); MONAT(d1); TAG(d1)); _
  1; 0)
```

Die Formulierung in VBA

- Die Formulierung in VBA ist etwas umständlicher, dafür aber besser lesbar

```
neuDifferenz = Year(d2) – Year(d1)
If d2 < DataSerial(Year(d2), Month(d1), Day(d1)) Then
  neuDifferenz = neuDifferenz – 1
End if
```

Zeitdifferenz in Monaten

- Das gleiche Problem tritt auch beim Rechnen mit Monaten auf. Wenn die Zeitdifferenz zwischen dem 25.1.1994 und dem 3.3.1994 als zwei Monate gelten soll, können Sie die ersete (einfachere) Formel anwenden. Wenn die Differenz erst ab dem 25.3.1994 zwei Monate betragen soll, müssen Sie die zweite Formel anwenden (fliessende Datumsgrenzen)

```
= (JAHR(d2)-JAHR(d1))*12 + MONAT(d2)-MONAT(d1)   'Monatsgrenze 1.
```

```
= (JAHR(d2)-JAHR(d1))*12 + MONAT(d2)-MONAT(d1) -
   WENN(d2>DATUM(JAHR)d2; MONAT(d2); TAG(d2)); 1; 0)
  'fliessende Monatsgr
```

Datum des nächsten Montags

Manchmal tritt das Problem auf, dass Sie ein beliebiges Datum haben und daraus das Datum des nächsten Monats (oder eines beliebigen anderen Wochentags) benötigen. Die VBA-Formel zur Berechnung des nächsten Montags sieht folgendermassen aus:

```
D = d + (9 - WeekDay(d)) Mod 7
```

Dazu ein Beispiel: Angenommen, d ist ein Mittwoch: Dann leifert `WeekDay` den Wert `4`. `(9-4) Mod 7` liefert `5`, es werden zum aktuellen Datum also fünf Tage hinzugezählt. Wenn d bereits ein Montag war, ändert sich das Datum nicht. Wenn Sie die Formel so ändern möchten, dass der nächste Dienstag, Mittwoch etc. berechnet werden soll, ersetzen Sie einfach die Zahl `9` durch `10, 11` etc.

Differenzen über Mitternacht

Die unmittelbare Berechnung der Differenz zweier reiner Zeiten über Mitternacht – etwa von 20:30 bis 6:40 – liefert einen negativen Wert. Dem kann mit einer Addition um 1 (entspricht 24 Stunden) abgeholfen werden:

```
= Wenn(z2<z1; 1+z2-z1;z2-z2)
```

Datumsdifferenz ausrechnen

- Im Makro wird die Differenz zwischen dem heutigem Datum und dem Datum, welches in Zelle A1 steht, ermittelt und am Bildschirm ausgegeben.

```
Sub DatumsdifferenzenAusrechnen()
Dim DatWert As Date
 DatWert = Range("A1").Value
 MsgBox CLng(Date - DatWert) & " Tage ist die Differenz"
End Sub
```

Prüfen, ob aktive Zelle ein gültiges Datum hat

```
Sub DatumPrüfen()
 If Not IsDate(ActiveCell.Value) Then
    MsgBox "Kein Datum!"
 End If
End Sub
```

Feiertage
In Vorlage: Holiday

MonatNr = Month(Now())

Int-, Fix-Funktionen
Gibt den ganzzahligen Anteil einer Zahl zurück.

Syntax
```
Int(Zahl)
Fix(Zahl)
```

Bemerkungen
Int und **Fix** entfernen beide die Nachkommastellen einer *Zahl* und geben den daraus resultierenden ganzzahligen Wert zurück.
Der Unterschied zwischen **Int** und **Fix** besteht darin, daß bei einem negativen Wert von *Zahl* **Int** die erste negative ganze Zahl zurückgibt, die kleiner oder gleich `Zahl` ist, während **Fix** die erste negative ganze Zahl zurückgibt, die größer oder gleich `Zahl` ist. **Int** wandelt zum Beispiel -8,4 in -9 um, während `Fix` -8,4 in -8 umwandelt.
Fix(*Zahl*) entspricht dem folgenden Ausdruck:
```
Sgn(Zahl) * Int(Abs(Zahl))
```

14.7 Syntaxzusammenfassung

VBA-Funktionen

Date	liefert das aktuelle Datum
Date = dz	verändert das Systemdatum
Time	leifert die aktuelle Zeit
Time = dz	verändert die Systemzeit
Now	liefert Datum und Zeit
Timer	liefert die Sekunden seit 00:00
DateValue()	wandelt die Zeichenkette in ein Datum um
DateSerial(jahr, mon, tag)	setzt die drei Werte zu einem Datum zusammen
Year(dz)	liefert Jahreszahl
Month(dz)	liefert Monat (1-12)
Day(dz)	liefert Tag (1-31)
WeekDay(dz)	liefert Wochentag (1-7 für So-Sa)
WorksheetFunction.WeekDay(dz, 2)	liefert Wochentag (1-7 für Mo-So)
WorksheetFunction.WeekDay(dz, 3)	liefert Wochentag (0-6 für Mo-So)
WorksheetFunction.Days360(dz1, dz2)	Tagesdifferenz in einem 360-Tage-Jahr
WorksheetFunction.Days360(dz1, dz2, False)	wie oben, europäischer Modus
TimeValue(z)	wandelt Zeichenkette in Zeit um
TimeSerial(st, min, sek)	setzt die drei Werte zu einer Zeit zusammen
Hour(dz)	liefert Stunde (0-23)
Minute(dz)	liefert Minute (0-59)
Secound(dz)	liefert Sekunde (0-59)
CDate(v)	Umwandlung ins *Date*-Format
CStr(dz)	Umwandlung in Zeichenkette
CSng(dz)	Umwandlung in einfach genaue Fliesskommazahl
CDbl(dz)	Umwandlung in doppelt genaue Fliesskommazahl
WeekdayName(n)	liefert Zeichenkette mit Wochentagsname (1 entspricht Montag, 7 Sonntag)
MonthName(n)	liefert Zeichenkette mit Monatsnamen
FormatDateTime(d, type)	liefert Zeichenkette mit Datum oder Zeit *(type=vbGeneralDate /vbLongDate / vbShortDate /*

vbLongTime / vbShortTime

Tabellenfunktionen (für die deutsche Excel-Version)

HEUTE()	aktuelles Datum
JETZT()	aktuelle Zeit
DATUM(jahr, mon, tag)	setzt die drei Werte zu einem Datum zusammen
DATWERT(z)	wandelt Zeichenkette in ein Datum um
JAHR(dz)	liefert Jahreszahl
MONAT(DZ)	liefert Monat (1-12)
TAG(dz)	liefert Tag (1- 31)
WOCHENTAG(dz)	liefert Wochentag (1-7 für So-Mo)
WOCHENTAG(dz, 2)	liefert Wochentag (1-7 für Mo-So)
WOCHENTAG(dz, 3)	liefert Wochentag (0-6 für Mo-So)
TAGE360(dz1, dz2)	Tagesdifferent in einem 360-Tage-Jahr
TAGE360(dz1, dz2, Falsch)	wie oben, europäischer Modus
ZEIT(st, min, sek)	setzt die drei Werte zu einer Zeit zusammen
ZEITWERT(z)	wandelt Zeichenkette in eine Zeit um
STUNDE(dz)	liefert Stunde (0-23)
MINUTE(dz)	liefert Minute (0-59)
SEKUNDE(dz)	liefert Sekunde (0-59)

14.8 Benutzerdefinierte Tabellenfunktionen (VBA)

- Die Berechnung einer benutzerdefinierten Tabellenfunktion ist in jedem Fall ungleich langsamer als die Verwendung einer in Excel vordefinierten Funktion.

- VBA sieht sich nicht veranlasst, bei einer Veränderung von VBA-Code die davon betroffenen Zellen automatisch neu zu berechnen. Die explizite Aufforderung zur Neuberechnung durch F9 funktioniert meistens, aber leider nicht immer ganz zuverlässig. In besonders hartnäckigen Fällen hilft es meistens, eine neue Zeile oder Spalte oberhalb bzw. links von den betroffenen Zellen in die Tabelle einzufügen und anschliessend wieder zu entfernen.

Kurzbeschreibung für den Funktionsassistenten

- Im Dialog ”Einfügen / Funktion” wird zu allen Funktionen eine kurze Beschreibung angezeigt. Dazu im Objektkatalog die Funktion markieren und mit der rechten Maustaste das Kommando ”Eigenschaften” auswählen. Der darauf erscheinende Dialog ”Elementoptionen” macht zwar einen etwas unfertigen Eindruck, erfüllt aber seine Aufgabe.

Funktionskategorien

- Wenn Sie eigene Funktionen in einer anderen Kategorie als ”Benutzerdefiniert” zuordnen möchten, müssen müssen Sie im Direktfenster eine Anweisung wie im folgenden Beispiel ausführen:

```
Application.MacroOptions Macro:=”Discount”, _
  Category:=14Discount”, Category:=14
```

- Die Funktion wird damit bleibend der Gruppe ”Finanzmathematik” zugeordnet(d.h. die Einstellung wird zusammen mit der Excel-Datei gespeichert). Die folgende Liste gibt die wichtigsten Kategoriennummern an:

Category Kategoriename

1	Finanzmathematik
2	Datum und Zeit
3	Math. & Zeit
4	Statistik
5	Matrix
6	Datenbank
7	Text
8	Logik
9	Information
14	Benutzerdefiniert (default)

Benutzerdefinierte Funktion in anderen Arbeitsmappen verwenden

- Sie müssen den Dateinamen angeben, in der die Funktion definiert ist – also: `= Funktion.xls!Discount(8,12)`.
- Alternativ können Sie in der aktuellen Arbeitsmappe mit ”Extras/Verweise” einen Verweis auf ”Function.xls” einrichten, dann ist eine Verwendung ohne den vorangestellten Dateinamen möglich.

Benutzerdefinierte Funktionen in Add-Ins

- Sie können eine Arbeitsmappe mit den Definitionen mehrerer eigener Funktionen in ein Add-In kompilieren.

Zellbereiche als Parameter

- Problematisch wird es, wenn als Parameter ein (womöglich aus Teilbereichen zusammengesetzter) Zellbereich übergeben wird – etwa A1:A3.
- Eine besondere Komplikation stellt der Umstand dar, dass ”A1:A3,C1:C3” in Excel üblicherweise einen Zellbereich meint, der aus den Teilbereichen A1:A3 und C1:C3 zusammengesetzt ist. Bei ”A1:A3;C1:C3” kann es sich aber genauso gut auch um *zwei* Argumente (für eine Funktion mit zwei Parametern) handeln! Wenn Zellbereihce unmittelbar in dieser Form angegeben werden, interpretiert Excel die Zeichen tatsächlich als zwei Argumente. Wenn dagegen aus Gründenn der Eindeutigkeit der gesamte Zellbereich geklammert wird (also ”(A1:A3;C1:C3)”) oder wenn der Zellbereich in einem Namen gespeichert wird, dann betrachtet Excel das Argument als zu *einem* Parameter gehörig.
- Aus diesem Grund ist die Programmierung von Funktionen, die mit beliebig zusammengesetzten Zellbereichen zurechtkommen sollen, ein wenig umständlich: Die Funktion `QuadSum` quadriert die Werte aller angegebenen Zellen. Dabei wird der Parameter der Funktion als `ParamArray` definiert, so dass beliebig viele Parameter übergeben werden dürfen. Für jeden dieser Parameter werden alle Zellen der Teilbereiche quadriert und in `result` summiert. Dank der Abfrage `TypeName(var)=”Range”` kommt `Quadsum` auch mit numerischen Parametern zurecht. `QuadSum(1;2;3)` liefert also 14.

```
Function QuadSum(ParamArray x())
  Dim var As Variant, result As Double
  Dim a As Range, c As Range
  For Each var In x()
    If TypeName(var) = ”Range” Then
      For Each a In var.Areas         'alle Teilbereiche
        For Each c In a.Cells         'alle Zellen je Teilbereich
          result = result + c ^ 2
        Next c
      Next a
    Else
```

```
      result = result + var ^ 2
    End If
  Next var
  QuadSum = result
End Function
```

Fehlerabsicherung

- Wenn Sie eine Funktion programmieren möchten, die nicht für Zellbereiche, sondern ausschliesslich für Einzelwerte konzipiert ist, sollten Sie eine falsche Parameterübergabe durch Sicherheitsabfragen ausschliessen. Die so abgesicherte *Discount*-Funktion sieht dann folgendermassen aus:

```
Function Discount(unitprice, pieces)
  On Error Resume Next
  If TypeName(unitprice) = "Range" Then
    If unitprice.Count > 1 Then Discount = CVErr(xlErrValue): Exit
Function
  End If
  If TypeName(pieces) = "Range" Then
    If pieces.Count > 1 Then Discount = CVErr(xlErrValue): Exit
Function
  End If
  If pieces >= 10 Then
    Discount = pieces * unitprice * 0.95
  Else
    Discount = pieces * unitprice
  End If
  If Err Then Discount = CVErr(xlErrValue)
End Function
```

- Mit **`CVErr(xlErrValue)`** wird ein **`#WERT!`-Fehler** als Ergebnis zurückgegeben. Mögliche Konstanten für `CVErr` finden Sie im Objektkagalog in der Konstantengruppe **xlCVError**.

Matixfunktionen

RGP (In Online-Hilfe) – gibt die Parameter eines linearen Trends zurück.

Die Volatile-Methode

```
Application.Volatile True
```

- Dadurch wird die Funktion jedes Mal neu berechnet, wenn irgendeine Zelle des Tabellenblatts neu berechnet wird. (Normalwerweise werden Funktionen nur dann neu berechnet, wenn sich dren Vorgängerzellen verändern. Das ist im Regelfall ausreichend und natürlich deutlich effizienter.)
- Der angezeigte Wert ändert sich jedes Mal, wenn irgendeine Zelle der Tabelle verändert oder neu berechnet wird.

14.9 Beispiele

Text in Ziffernwerte umwandeln

- Im Bankbereich, um Zahlen in Text umzuwandeln. Wandelt Zahlen in Text um, beispielsweise 12,34 zu: ---- eins zwei Komma drei vier ----

```
Function NumberToText(x)
  Dim i As Integer, result As String, character As String, lastchar
As Long
  Dim digit(9) As String
  digit(0) = ”Null”
  digit(1) = ”Eins”
  digit(2) = ”Zwei”
  digit(3) = ”Drei”
  digit(4) = ”Vier”
  digit(5) = ”Fünf”
  digit(6) = ”Sechs”
  digit(7) = ”Sieben”
  digit(8) = ”Acht”
  digit(9) = ”Neun”

  If IsEmpty(x) Then
    NumberToText ””
    Exit Function
  End If
  If x >= 10000000000# Or x <= -10000000000# Then
    NumberToText = ”Zahl zu groß oder klein”
    Exit Function
  End If
  If x < 0 Then
    result = ”Minus ”
    x = -x
  End If
  x = Format$(x, ”0.00”)
  x = Space(13 - Len(x)) + x
  If Right(x, 3) = ”.00” Then
    lastchar = 10
  Else
    lastchar = 13
  End If
  For i = 1 To lastchar
    character = Mid(x, i, 1)
    If character >= ”0” And character <= ”9” Then
      result = result + digit(Val(character)) + ” ”
    ElseIf character = ”.” Then
      result = result + ”Punkt ”
    End If
  Next i
  NumberToText = ”---- ” + Trim(result) + ” ----”
End Function
```

Zufallszahlen

```
Sub Zufallszahlen()
Dim Zelle As Object
For Each Zelle In Selection
   Zelle.FormulaR1C1 = ”=INT(RAND()*100)”
 Next Zelle
End Sub

In Zelle:
= Ganzzahl(Zufallszahl()*100)
```

Zufallszahlen als Matrixformel

```
Selection.FormulaArray = "=INT(RAND()*100)"
```

Zufallszahlen zwischen 1 und 49

Dim zufallszahl As Byte
Zufallszahl = Int(Rnd * 49) + 1

14.10 Formeln und Funktionen

Formel einer Zelle zuweisen

```
Range("A10").Formula = WorksheetFunction.Sum(Range("A1:A9"))
```

- Da es sich um eine Funktion handelt, müssen Sie die Eigenschaft "Formula" verwenden, um der Zelle die Funktion "Sum" zuzuweisen.

Weitere wichtige Worksheet-Functionen

- Max, Min, Average

- Weitere wichtige Worksheet-Functionen sind **`Max`**, **`Min`**, **`Average`** welche den höchsten, den kleinsten und den Durchschnittswert einer Liste ermitteln.

- Mittelwert über Inputbox ermitteln

```
Dim Bereich As Range
  On Error Resume Next
  Set Bereich = Application.InputBox(Prompt:="Markieren Sie den
Zellbereich:", Type:=8)
  MsgBox WorksheetFunction.Average(Bereich)
```

Mittelwert eines Zellbereichs herausfinden

```
Sub Mittelwert
Dim Bereich As Range
Dim Mittelwert As Singel
  Set Bereich = ActiveSheet.Range("A1:C1")
  Mittelwert = Application.WorksheetFunction.Average(Bereich)
  MsgBox Promt:="Der Mittelwert ist: " & Mittelwert
End Sub
```

FormelZellenFärben

```
Selection.SpecialCells(xlCellTypeFormulas, 23).Select
 For Each Zelle In Selection
  Zelle.Interior.ColorIndex = 15
 Next Zelle
```

Text in Spalten aufteilen

Vorher:

Fratton, Mario

Nachher:

Fratton	Mario

```
Columns("B:B").Insert Shift:=xlToRight
Do While ActiveCell.Value <> ""
  i = InStr(ActiveCell.Value, ",")                    - OK
  i2 = Len(ActiveCell.Value)                          - OK
  Nachname = Left(ActiveCell.Value, i - 1)            - OK
```

```
  Vorname = Mid(ActiveCell.Value, i + 2, i2 – i)      - OK
```

- `Mid(string, start[, length])`

Daten bereinigen nach Datentransfer

```
Dim Zelle As Range
For Each Zelle In ActiveSheet.UsedRange
    With Zelle
    If .HasFormula = False Then
        .Value = Application.WorksheetFunction.Clean(.Value)
    End If
    End With
Next Zelle
```

- `Clean` entfernt alle nichtdruckbaren Zeichen

Zahlen als Werte und nicht als Text erkennen

```
Dim Zelle As Range
For Each Zelle In Selection
  Zelle.Value = Zelle.Value*1
Next Zelle
```

Schriftfarbenwechsel durchführen

```
For Each Zelle In Selection
  Zelle.Characters(1, 3).Font.Color = vbRed
Next
```

Die Anzahl Wörter im markierten Bereich ermitteln

```
Dim Bereich As Range
Dim Zelle As Object
Dim s As String
Dim l As Long

Set Bereich = Selection
For Each Zelle In Bereich
  s = Trim(Zelle.Text)
  Do While InStr(s, ” ”) > 0
    l = l + 1
    s = Trim(Right(s, Len(s) - InStr(s, ” ”)))
  Loop
   l = l + 1
Next Zelle
MsgBox ”Es wurden im markierten Bereich ” & _
Selection.Address & Chr(13) & l & ” Wörter gefunden!”
```

- `Trim` entfernt sowohl die Leerzeichen am Beginn der Zelle als auch die Leerzeichen am Ende der Zelle. Dies ist die erste Voraussetzung, die erfüllt sein muss, damit die folgenden Schritte zuverlässig ablaufen. Nun setzen Sie eine `Do Whil`-Schleife auf, in der Sie die einzelnen Wörter ermitteln. Dabei setzen Sie die Funktion ”InStr” ein, um das Auftreten des ersten Leerzeichens in der Zelle (zwischen zwei Wörtern) aufzuspüren. Innerhalb der Schleife zerlegen Sie den Text in die einzelnen Worte und erhöhen den Zähler ”l” bei jedem gefundenen Wort. Die Schleife wird so lange durchlaufen, bis die Funktion ”InStr” den Wert 0 zurückliefert; damit ist das Ende der Zelle erreicht. Zuletzt geben Sie in einer Meldung die ermittelte Anzahl der Wörter aus.

Das Auftauchen eines Zeichens im markierten Bereich ermitteln

```
Dim Bereich As Range
Dim Zelle As Object
Dim i As Integer
Dim i2 As Integer
Dim s As String
i = 0
 s = InputBox(”Geben Sie das Zeichen ein, welches Sie zählen
möchten”)
 If s = ”” Then Exit Sub
    For Each Zelle In Selection
       i2 = InStr(1, Zelle.Value, s)
       While i2 <> 0
         i = i + 1
         i2 = InStr(i2 + 1, Zelle.Value, s)
       Wend
    Next Zelle
 MsgBox ”Das Zeichen ” & s & ” trat im Bereich ” _
   & Selection.Address & ” genau ” & i & ” Mal auf!”
```

- In einer `For Each`-Schleife durchlaufen Sie den markierten Bereich. In der Integer-Variablen `i2` zählen Sie das Vorkommen des gesuchten Zeichens in der Zelle. Danach setzen Sie eine zweite Schleife auf. Beim Eintritt in die `While`-Schleife wurde bereits ein Vorkommen des gesuchten Zeichens ermittelt. Aus diesem Grund zählen Sie die Integer-Variable `i` um einen Zähler hoch. Jetzt setzen Sie den Mauszeiger ein Zeichen in der Zelle nach forn und suchen nach dem nächsten Vorkommen, indem Sie die Funktion `InStr` anwenden. Sobald in der `While`-Schleife kein Vorkommen des gesuchten Zeichens mehr ermittelt werden kann, meldet die Funktion `InStr` den Wert `0`. Dann ist das Abbruchkriterium der `While`-Schleife erfüllt und die Bearbeitung kann mit der nächsten Zelle im markierten Bereich weitergehen.

- `MsgBox Time & ” Uhr”`
- `MsgBox Date`

14.11 Diverses

Diverses

- F9 zur Neuberechnung der Arbeitsmappe
- vbKonstanten
  xlKonstanten
  `Application.Pi` (Weder noch)
- Function **IsMissing** ermittelt, ob ein optionales Argument übergeben wurde.

Syntax eine Funktinonsprozedur

```
Function Functionname [(Argumentenliste)]
  Anweisungen
  Functionname = Ausdruck    – Funktionswert
End Function

Function Functionname(Argument1, Argument2, Argument3)
Functionname = Ausdruck
End Function
```

```
Function Functionname [(Argumentenliste)] As Datentyp
```

Syntax einer Sub-Prozedur

```
Sub Prozedurname [(Argumentenliste)]
  Anweisungen
End Sub
```

Werte und Formeln in einen Bereich eintragen

```
Sub FormelBeobachten()
  Range(”C2:C6”).Select
  Selection.Formula = 100                  - Alle Zellen 100
  ActiveCell.Formula = 0                   - Aktive Zelle 0
  ActiveCell.Offsetz(-1, 0).Formula = 1    - Eine Zelle oberhalb
                                             aktiver Zelle = 1
  Selection.Formula = ”=C1*5”              - Zelle jeweils oberhalb
                                             * 5. Erste markierte
                                             Zelle ist C2
  MsgBox ActiveCell.Value                  - Wert
  MsgBox ActiveCell.Formula                - =C1*5
  MsgBox ActiveCell.FormulaR1C1            - =R[-1]C*5)
End Sub
```

Summen Berechnen

```
Sub SummenBerechnen
Dim neuBereich As Range
Dim neuGesamt As Range
  Set neuBereich = ActiveCell.CurrentRegion
  Set neuGesamt = neuBereich.Offset(neuBereich.Rows.Count).Row(1)
  ’Bereich, der die Summen aufnimmt
  neuGesamt.Cells(1) = neuBereich.Columns(1).Address
  ’Absolute Adresse der ersten Spalte
  neuGesamt.Cells(1) = neuBereich.Columns(1). Adresse(False, False)
  ’relative Adresse der ersten Spalte
  neuGesamt.Formula = _
    ”Sum(” & neuBereich.Columns(1).Adress(False, False) & ”)”
End Sub

???????Stimmt das da oben???????????
```

FormluaLocal, FormulaR1C1Local, Formula, FormulaR1C1

- **`FormulaLocal`** und **`FormulaR1C1Local`**: Die Eigenschaften liefern die Formel der Zelle in der A1- oder in der Z1S1-Schreibweise (siehe unten). Bei leeren Zellen wird eine leere Zeichenkette, bei Formeln mit Konstanten der Wert der Konstante zurückgegeben. Wenn A5 die Formel `=SUMME(A1:A4)` enthält, gibt `[A1].FormulaLocal` die Zeichenkette = SUMME(A1:A4) zurück. [A1]FormulaR1C1Local liefert `=SUMME(Z[-4]S:Z[-1]S)`.

- **`Formula, FormulaR1C1`**: Die beiden verwandten Eigenschaften leifern die ins Englische übersetzte Formels in der `A1`- oder in der `R1C1`-Schreibweise. `[A1].Formula` gibt also die Zeichenkette `=SUM(A1:A4)` zurück. `[A1].FormulaR1C1` liefert `=SUM(R[-4]C:R[-1]C`

?Range(”A3”).Formula

=SUM(A1+A2)

?Range(”A3”).FormulaLocal
=SUMME(A1+A2)

Z1S1-Notation

- Extras / Optionen / Z1S1-Bezugsart
- `=Z(-2) S(-1)`
- Formeln bleiben immer unverändert
- Normal ist A1-Notation

- `Selection.FormulaR1C1 = ”=R[-1]C”`
- `Selection.Formula = ”=C2”`
- FormulaR1C1 = Z1S1Formel
- FormulaR1C1Local = Z1S1 LokaleFormel

Zeichen um Zeichen mit grösserer Schrift versehen

```
If IsEmpty(ActiveCell.Value) Or ActiveCell.HasFormula Then Exit Sub
If IsNummeric(ActiveCell.Value) Then Exit Sub
For i = 1 To ActiveCell.Characters.Count
  ActiveCell.Characters(i, 1).Font.Size = 9 + i
Next
```

Runden mit Excel

```
Sub Runden()
Dim Zelle As Object
  For Each Zelle In Selection
    If Zelle.Value = ”” Or Zelle.Value = 0 Then
    Else
      On Error Resume Next
      Zelle.Value = _
        CDec(Format(Application.Round(Zelle.Value, 2), ”0.00”))
    End If
  Next Zelle
End Sub
CDec sorgt dafür, dass das Ergebnis auch als Zahlenwert erhalten
bleibt.
```

Funktion Mwst

```
Public Function MWSt(brutto As Single)
MWSt = brutto / 166 * 16
End Function
```

Funktion Add

```
Function Add(x, y)
Add = x + y
End Function
= Add(12; 7)
```

Datum- / Uhrzeit-Eingabe

- `#1. Jan 1999#`

Vordefinierte Funktionen

Sqr

- Ermittelt Quadratwurzel einer Zahl `v = Sqr(x)`

Len

- Ermittelt Länge einer Zeichenkette `v = Len(x$)`

Int (=Ganz)

- Wird ein nummerischer Wert übergeben. Als Funktinswert liefert Int wieder ein nummerischer Wert. Den ganzzahligen Anteil der übergebenen Zahl. Die grösste ganze Zahl, die kleiner oder gleich "x" ist.

Left

```
v = Left(x$, n)
```

Right

```
v = Right(x$, n)
```

Mid

```
v = Mid (x$, n [, m])
```

InStr

```
v = InStr([n,] x$, y$)
```

- `n` = Suchstart
- `x$` = String, der durchsucht wird
- `y$` = String, der gesucht wird

14.12 Variable einer anderen Prozedur oder Funktion übergeben

- Der Inhalt einer Variable soll einer anderen Prozedur übergeben werden. VBA stellt dafür einen eigenen Mechanismus zur Verfügung
- Sie können einer Prozedur beliebig viele Argumente übergeben
- Beim Aufruf einer Prozedur kann auch eine Variable übergeben werden. Dann ist das Lokalistätsprinzip jedoch durchbrochen.

Funktions-Beispiel 1

```
Public Function  MWSt_okay_aufruf()
  Call MWSt_okay(23.4)
End Function

Public Sub MWSt_okay(brutto As Single)
Dim MWSt As Single, netto As Single
  MWSt = brutto / 116 * 16
  Netto = brutto – MWSt
  Debug.Print "MWSt: "; MWSt
  Debug.Print "Netto: ": netto
End Sub
```

Funktions-Beispiel 2

```
Public Function Referenz_aufruf()
Dim y As Single, z As Single
  Call Referenz(23.4, y, z)
  Debug.Print "MWSt: " ; y
  Debug.Print "MWSt: "; z
End Function

Public Sub Referenz(brutto As Single, MWSt As Single, _
Netto As Single)
  MWSt = brutto / 116 * 16
```

```
  Netto = brutto – MWSt
End Sub
```

- Jede Änderung einer der in der Argumentenliste von Referenz aufgeführte Variable wirkt auf die korrespondierende Variable der aufrufenden Prozedur zurück: Jede Veränderung von MWSt führt zur gleichen Veränderung bei `y` und jede Veränderung von Netto zur entsprechenden Beeinflussung von `z`.
- Der Name ist bedeutungslos. VBA interessiert sich ausschliesslich für die Reihenfolge.

Prozedur-Beispiel 1

- ```
  Sub Test(Einkaufspreis As Single, Verkaufspreis As Single)
  Call Test(200, 300)
  ```

Prozedur-Beispiel 2

- ```
  Call MWSt(20 + 3.4)
  Prozedur-Beispiel 3
  ```
- ```
  x = 20
  Call MWSt(x + 3.4)
  ```

Prozedur-Beispiel 4

```
Sub TestprocAufruf
  x = 50
  Call Textproc(x)
  Deebugprint(x)
End Sub

Sub Testproc(zahl)
  zahl = zahl + 50
End Sub
```

- `x` ist die korrespondierende Variable
- Wird jene Variable in der Argumentenliste der aufgerufenen Prozedur verändert, in der das Argument übernommen wird, wird dadurch auch die korrespondierende Variable der aufrufenden Prozedur verändert.
- Diesen Mechanismus nennt man Variablenübergabe als Referenz. Er kann eingesetzt werden, um es einer aufgerufenen Prozedur zu ermögichen, Informationen an die aufrufende Prozedur *zurück*zugeben.

Sie können auch Stringargumente übergeben

```
Call Stringtest(”Hallo”)

a$ = ”Hallo”
Call Stringtest(a$)

Sub Stringtest(s$)
Oder Sub Stringtest(s As String)

Call Stringtest(a$ + ”x” + b$ + ”y”)
```

Andere Reihenfolge mit benannten Argumenten

Prozedurname Variablenname1:= Wert, Variablennaem2:= Wert

- Syntax ist ohne Call und ohne Klammer
```

Optionale Argumente bei Makroprozedur

```
Sub Test(Optional Einkaufspreis As Variant, Optional Verkaufspreis
As Variant)
Call Test(, 300)
```

Optionale Parameter bei Funktionsprozedur (mit Standardwerten)

```
Function Zufall(Optional Mittelwert = 0.5, Optional _
Bereich = 0.5, Optional Runden = False)
```

Übergabe als Referenz ausdrücklich festlegen mit ByRef

```
Sub Testproc(ByRef Zahl As Single)
```

Eine Variable ”als Wert” übergeben

- Da die Abschottung der Variable verschiedener Prozeduren durchbrochen wird, könnte es ungewollte Veränderungen geben.

```
Call Wert((x))
Sub Wert(Zahl As Singel)
```

- Wird in der Prozedur Wert der Inhalt von `x` verändert, z.B. durch die Zuweisung zahl = 10, bleibt die Variable `x` der aufrufenden Prozedur diesmal unbeeinflusst. Die Klammerung schützt `x` vor jeder Veränderung.

Beispiel: Referenz und Wert Übergabe

```
Public Funktion Wert_aufruf()
Dim x As Single, y As Single
  x = 10
  y = 20
  Call Wert (x, (y))
  Debug.Print x
  Debug.Print y
End Function

Public Sub Wert(Zahl1 As Single, Zahl2 As Single)
  Zahl1 = 0
  Zahl2 = 0
End Sub
```

Direktbereich:
```
0
20
```

Private und Statische Prozeduren und Funktionen

```
[Public / Private] [Static] Sub Prozedurname[(Argumentenliste)]
[Public / Private] [Static] Function Prozedurname[(Argumentenliste)]
As Typ
```

- Private
Prozedur kann nur von Prozeduren des gleichen Moduls aufgerufen warden.
- Static
Der Inhalt aller lokalen Prozedurvariablen bleiben bis zum nächsten Aufruf unverändert

Statische Funktion: Beispiel 1

```
Public Function Statisch_aufruf()
  Debug.Print Statisch(5)  - 5
  Debug.Print Statisch(2)  - 7
  Debug. Print Statisch(8) - 15
End Function

Public Static Function Statisch(Zahl As Single)
Dim Summe As Single
  Summe = Summe + Zahl
  Statisch = Summe
End Function
```

14.13 Benutzerdefinierte Funktionen

Funktion Benotung

```
Function Benotung(r)
    Select Case r.Value
        Case Is = 6: Benotung = ”Sehr gut”
        Case Is = 5: Benotung = ”Gut”
        Case Is = 4: Benotung = ”Befriedigend”
        Case Is = 3: Benotung = ”Ausreichend”
        Case Is = 2: Benotung = ”Mangelhaft”
        Case Is = 1: Benotung = ”Ungenügend”
        Case Else: Benotung = ”keine gültige Zensur”
    End Select
End Function
```

Enthält eine Bestimmte Zelle eine Formel

```
Function IstFormel(r)
    IstFormel = False
    If Left(r.Formula, 1) = ”=” Then IstFormel = True
End Function
```

Initialen aus dem Namen erstellen

Es werden die Anfangsbuchstaben der Namen ausgewertet und zusammengebastelt.

```
Function Initial(str As String) As String
    Dim Count As Integer
    str = ” ” & Application.Trim(str)
    For Count = 2 To Len(str)
        If Mid(str, Count - 1, 1) = ” ” Then _
          Initit = Initit & Mid(str, Count, 1)
    Next Count
End Function
```

Formeln ausgeschrieben anzeigen

```
Function FormelInText(r)
    Application.Volatile
    FormelInText = r.FormulaLocal
End Function
```

Nur Zellen mit Fettdruck addieren

```
Function FormatAddieren(r As Range)
```

```
    For Each r In r.Cells
        If IsNumeric(r) Then
            If r.Font.Bold = True Then
                FormatAddieren = FormatAddieren + r.Value
            End If
        End If
    Next r
End Function
```

Zeit in Minuten anzeigen

```
Function ZeitInMinuten(EZeit As Date) As Integer
  ZeitInMinuten = EZeit * 24 * 60
End Function
```

Zeit in Stunden anzeigen

```
Function ZeitInStunden(EZeit As Integer) As Date
  ZeitInStunden = EZeit / 24 / 60
End Function
```

Hintergrundfarbe auslesen

- Stellen Sie sich vor, Sie arbeiten mit gefärbten Zellen. Dafür definieren Sie eine Reihe von Farben, die in Ihrer Tabelle gültig sein sollen. Bei Verwendung von anderen Farben soll der Text `Keine gültige Farbe!` ausgegeben werden.

Function Farbe(r As Range)

```
Dim FarbeG As String
    Application.Volatile
    Select Case r.Interior.ColorIndex
     Case 1
        FarbeG = ”Schwarz”
     Case 2
        FarbeG = ”Weiß”
     Case 3
        FarbeG = ”Rot”
     Case 4
        FarbeG = ”Grün”
     Case 5
        FarbeG = ”Blau”
     Case Else
        FarbeG = ”keine gültige Farbe!”
    End Select
    Farbe = FarbeG
End Function
```

Zahlenformat ermitteln

```
Function ZahlenformatErmitteln(r)
 Application.Volatile
 FormatA = r.NumberFormatLocal
End Function
```

Umrechen der zurückgelegten Meter und der benötigten Zeit in Km/h

```
Function KmProStunde(Meter, Sekunden)
     KmProStunde = Meter / Sekunden * 3.6
End Function
```

Adresse mit höchstem Wert im Bereich ermitteln

```
Function MaxWertAdresse(r As Range)
Dim rng As Range
Dim MaxWert As Double

    MaxWert = WorksheetFunction.Max(r)
    MaxWertAdresse = ""
    For Each rng In r
        If rng.Value = MaxWert Then
           MaxWertAdresse = MaxWertAdresse & rng.AddressLocal & "; "
        End If
    Next
End Function
```

- Den grössten Wert innerhalb der Markierung ermitteln Sie über die Funktion `Max`. Für den Fall, dass es gleich mehrere gleiche Maximalwerte im markierten Bereich gibt, speichern Sie alle Adressen mit dem Verketttungsoperator & einfach hinten an.

Adresse mit niedrigstem Wert im Bereich ermitteln

```
Function MinWertAdresse(r As Range)
Dim rng As Range
Dim MinWert As Double

    MinWert = WorksheetFunction.min(r)
    MinWertAdresse = ""
    For Each rng In r
        If rng.Value = MinWert Then
           MinWertAdresse = MinWertAdresse & rng.AddressLocal & "; "
        End If
    Next
End Function
```

- Den niedrigsten Wert innerhalb der Markierung ermitteln Sie über die Funktion `Min`. Kommt der Minimalwert mehrmals im markierten Bereich vor, geben Sie alle Zelladressen über die IF-Abfrage in Verbindung mit dem Verkettungsoperator & bekannt

14.14 Modulare Funktionen

- Neben Funktionen, die Sie speziell für das Tabellenblatt schreiben, können Sie auch Funktionen programmieren, die Sie innerhalb der Entwicklungsumgebung im Zusammenspiel mit Makros einsetzen. Diese Funktionen sind dann ratsam, wenn sie in mehreren Makros gebraucht werden. Anstatt denselben Programmcode mehrmals zu erfassen, schreiben Sie einmal eine Funktion dazu und rufen diese aus den Makros eifach auf. Diese Programmierweise ist übersichtlich, pflegeleicht und macht Spass.

Dateien in einem Verzeichnis zählen

- Funktion

```
Function DateienZählen(str) As Long
Dim DatNam As String
Dim n As Long
    DatNam = Dir$(str & "\*.*")
    Do While Len(DatNam) > 0
            n = n + 1
            DatNam = Dir$()
    Loop
    DZ = n
End Function
```

- Die Funktion erwartet als Eingabe den Namen des Verzeichnises, auf welches Sie zugreifen möchten. Als Ergebnis liefert die Funktion Ihnen im Datentyp "Long" die Anzahl der ermittelten Dateien. Wenn Sie nur bestimmte Dateien gezählt haben möchten, können Sie die obige Funktion abändern, indem Sie die Zeichenkettenfolge `DatNam = Dir$(str & "\*.*")` beispielsweise in `DatNam = Dir$(str & "\*.xls")` ändern.
- Jetzt fehlt nur noch das Makro, welches der Funktion das Verzeichnis übergibt und die Rückmeldung der Funktion auswertet.

- Makro

```
Sub ZählenDateien()
Dim i As Long
  i = DateienZählen ("c:\temp\")
  MsgBox i
End Sub
```

Prüfen, ob eine bestimmte Datei existiert

- Funktion

```
Function DateiVorhanden(str As String) As Boolean
   DateiVorhanden = False
   If Len(str) > 0 Then DateiVorhanden = (Dir(str) <> "")
   Exit Function
End Function
```

- Die Prüfung, ob überhaupt eine Zeichenfolge an die Funktion übergeben wurde, erfolgt über die Funktion `Len`. Wird eine Länge von 0 gemeldet, wurde überhaupt keine Zeichenfolge an die Funktion übergeben. Wenn ja, entspricht diese in jedem Fall einer Grösse >0. Die Funktion `Dir` versucht nun auf die Datei zuzugreifen. Ist die Datei nicht vorhanden, meldet die Funktion eine Leerfolge zurück. Damit wird der Datentyp

`Boolean` mit dem Wert `False` an das aufrufende Makro zurückgemeldet. Im anderen Falle liefert die Funktion den Wert `True` zurück.

- Makro

```
Sub DateiDa()
  Dim bln As Boolean
  bln = DateiVorhanden(”C:\eigene Dateien\Mappe1.xls”)
  MsgBox bln
End Sub
```

Prüfen, ob eine bestimmte Datei geöffnet ist

- Funktion

```
Function DateiSchonGeöffnet(ByVal str As String) As Boolean
    On Error GoTo fehler
    DateiSchonGeöffneted = True
    Windows(str).Activate
    Exit Function
fehler:
    DateiSchonGeöffneted = False
End Function
```

- Die obige Funktion erwartet als Parameter den Namen der Datei. Danach wird einfach mal versucht, die entsprechende Datei zu aktivieren. Schlägt dies fehl, dann sorgt die `On Error`-Anweisung dafür, dass die Funktion nicht mit einer Fehlermeldung abbricht. In diesem Fall wird direkt zum Fehlerabschnitt gesprungen. Dort wird die Variable ”DateiSchonGeöffnet” auf den Wert `False` gesetzt. Im Fall dessen, dass die Aktion erfolgreich war, wird die Funktion über die Anweisung `Exit Function` direkt verlassen. Dadurch, dass Sie zu Beginn die Variable ”DateiSchonGeöffnet” auf den Wert `True` gesetzt haben, können Sie direkt aus der Funktion springen. Das aufrufende Makro sieht wie folgt aus:

- Makro

```
Sub DatGeöffnet()
  b = DateiSchonGeöffnet(”c:\eigene Dateien\mappe1.xls”)
  MsgBox b
End Sub
```

Prüfen, ob ein Add-In eingebunden ist

- Funktion

```
Function AddInEingebunden(ByVal str As String) As Boolean
Dim add As Object
    Set add = AddIns(str)
    If add.Installed = True Then
        AddInEingebunden = True
    Else
        AddInEingebunden = False
    End If
End Function
```

- Die Eigenschaft `Installed` liefert den Wert `True`, wenn das entsprechende Add-In eingebunden ist.

- Makro

```
Sub AddInDa()
Dim b As Boolean
    b = AddInEingebunden(”Solver”)
    If b = False Then
    MsgBox ”Solver Add-In ist nicht installiert.”
    Else
    MsgBox ”Solver Add-In ist installiert.”
    End If
End Sub
```

Dokumenteigenschaften einer Arbeitsmappe ermitteln

- Die Funktion ermittelt diverse Dokumenteigenschaften einer Arbeitsmappe. Dabei wird der Funktion der Dateiname sowie eine Eigenschafts-Nummer übergeben, durch die die Funktion dann die entsprechenden Informationen zur Verfügung stellt.
- Die einzelnen Informationen und die dazugehörigen Eigenschafts-Nummer entnehmen Sie folgender Tabelle:

Eigenschfts-Nummer	**Beschreibung**
0	Dateiname mit Pfad
1	Nur Pfad
2	Nur Dateiname
3	Dateityp
4	Dateigrösse in Byte
5	Erstellt am
6	Letzte Änderung am
7	Letzter Zugriff am

- Makro

```
Sub DokumentEigenschaften()
  MsgBox ZeigeDateiEigenschaften(”c:\Eigene Dateien\Mappe1.xls”, 7)
End Sub
```

- Funktion

```
Function ZeigeDateiEigenschaften(Dateiname, EigenschaftsNr As Byte)
On Error Resume Next
Dim fso As Object
Dim tmp As String
  Set fso = CreateObject(”Scripting.FileSystemObject”)
  With fso.GetFile(Dateiname)
    Select Case EigenschaftsNr
      Case Is = 0: tmp = .Path
      Case Is = 1: tmp = Mid(.Path, 1, Len(.Path) - Len(.Name))
      Case Is = 2: tmp = .Name
      Case Is = 3: tmp = .Type
      Case Is = 4: tmp = .Size
      Case Is = 5: tmp = CDate(.DateCreated)
      Case Is = 6: tmp = CDate(.DateLastModified)
      Case Is = 7: tmp = CDate(.DateLastAccessed)
      Case Else
        tmp = ”Ungültige EigenschaftsNr!”
    End Select
  End With
  ZeigeDateiEigenschaften = tmp
```

```
End Function
```

- Erstellen Sie im ersten Schritt einen Verweis auf das FileSystemObject, um damit die Informationen bezüglich der Arbeitsmappe zu erlangen. Danach werten Sie die übergebene Eigenschaften-Nummer in einer `Select Case`-Anweisung aus.

Bedingte Formatierung mit mehr als drei Farben

- Reicht Ihnen die Möglichkeit nicht, mit drei unterschiedlichen Farben zu arbeiten, schreiben Sie eine Funktion, welche die Formatierung Ihrer Daten automatisch vornimmt.

- Funktion

```
Function Muster(Zelle)
Dim i As Integer
  Select Case Zelle
    Case Is < 10: i = 3
    Case 11 To 20: i = 4
    Case 21, 23, 24: i = 6
    Case 22: i = 7
    Case Is > 25: i = 8
    Case Else: i = 2
  End Select
  With Selection.Interior
    .ColorIndex = i
    .PatternColorIndex = xlAutomatic
  End With
 End Function
```

- Makro

```
Sub Farbenspiel()
Dim i As Integer
  Range(”A1”).Select
  For i = 1 To ActiveSheet.UsedRange.Rows.Count
    Muster (ActiveCell)
    ActiveCell.Offset(1, 0).Select
  Next i
End Sub
```

- Ereignis

```
Private Sub Worksheet_Change(ByVal Target As Range)
  If Target.Column = 1 Then Farbenspiel
End Sub
```

Target heisst wenn ichs im Wörterbuch richtig verstanden habe ”Ziel” (wie Destination)

15 Fehlersuche, Fehlerabsicherung

- So wie jeder Fehler eine Nummer besitzt (`Err.Number`), hat auch jeder Fehler eine Beschreibung. Diese kann direkt gemeldet werden:

- `MsgBox Err.Description`

- Jeder Feher hat auch eine Quelle (`Err.Number`), eine Hilfenummer (`Err.HelpNumber`) und eine zugehörige Hilfedatei (`Err.HelpFile`). Diese können alle angezeigt werden. Nun kann die Fehlernummer abgefragt werden und eine gewünschte Aktion ausgeführt werden (Meldung des Fehlers, Speicherung …). Die Prozedur kann nun beendet werden oder mit `Resume` fotgesetzt werden.
- Soll der Inhalt der Variablen `Err` gelöscht werden, so kann dies mit `Err.Clear` geschehen
- Soll eine bestimmte VBA-Fehlermeldung angezeigt werden, so muss man die Fehlernummer wissen. Die Nummer 11 ist beispielsweise Division durch Null. Also gibt: `Err.Raise(11)` diese entsprechende Meldung

…

- ```
  Exit Sub
  Ende:
  If Err.Number = 11 Then
    MsgBox ”Bitte nicht durch Null teilen”
    Resume Next
  Else
    MsgBox Err.Description
  End If
  ```

- Sie können aber auch die zugehörige `Err.HelpFile`-Datei und die darin befindliche `Err.HelpContext` anzeigen oder aufrufen lassen. Für den Fehler mit der Nummer 11 (Division durch 0) lautet dies beispielsweise:

- ```
  If Err.Number = 11 Then
    MsgBox ”Bitte nicht durch Null teilen.” _
      & vbCr & vbCr & _
    ”Sie finden Hilfe in ” _
      & vbCr & vbCr & _
    Err.HelpFile & vbCr & ”unter der Nummer ” & Err.HelpContext
  ```

---[Scann einfügen]---

- Stop unterbricht das Makro

Fehlerverarbeitung

Variante 1

Sie tun nichts Besonderes. Sie geben sicherlich eine Fehlermeldung aus, vielleicht sogar eine verständlichere als die Systemmeldung. Ansonsten machen Sie nichts. Die jeweilige Sub wird dann einfach beendet. Vorteil: Für den Anwender läuft das Programm auf jeden Fall weiter, es ist nicht ”abgestürzt”. Es gibt zwar einen Fehler aus, bietet aber danach die Möglichkeit zum Weiterfahren.

```
Fehler:
  If Err.Number = 48 Then
    MsgBox ”Die Datei trouble.dll befindet sich nicht auf Ihrem
Rechner!”
  Else
    MsgBox Err.Description
  End If
```

Variante 2
Sie benutzen den Befehl `Resume`:

```
Fehler:
  MsgBox Err.Description
  Resume
```

Variante 3
Sie benutzen den Befehl `Resume Next`

- Grundsätzlich: Jede halbwegs komplexe Prozedur sollte eine On Error GoTo-Anweisung haben. Ich tendiere dazu, ein schlichtes MsgBox Err.Description einzuflechten und ansonsten gar nichts zu tun. Es gibt aber auch Fälle, in denen eine sinnvolle Behebung von Fehlern möglich ist oder wo zumindest bessere Fehlermeldungen ausgegeben werden können.

Noch ein Tipp zum Schluss: Aktivieren Sie die Fehlerbearbeitung währnd der Entwicklungs erst ganz am Ende. Sie können die Statements jederzeit einbauen, es genügt, die On-Error-Anweisung auszukommentieren.

Kleinere Fehler beheben

- Es empfiehlt sich, in dem Objektkatalog nachzuschauen, welche Eigenschaften und Methoden ein Objekt zur Verfügung stellt.
- Informieren Sie sich über die richtige Syntax eines Befehls, indem Sie über die F1-Taste in die Online-Hilfe wechseln und dort nachschauen.
- Sollten Sie bei der Suche nach Fehlern nicht weiterkommen, können Sie im Internet recherchieren.

Resume

- Fehlerhafte Zeile wird erneut ausgeführt. Resume schreibt man z.B. am Ende der Fehlerbehandlungsroutine, um die fehlerhafte Zeile erneut auszuführen.

Resume Next

- Nächste Zeile (nach Fehler) wird ausgeführt. Resume Next schreibt man z.B. am Ende der Fehlerbehandlungsroutine, um nach der fehlerhaften Zeile weiter zu fahren.

Geltungsbereich von Resume und Resume Next

- Die beiden Befehle `Resume` und `Resume Next` gelten immer nur für die Prozedur, in der sie eingesetzt werden. Wenn ein Fehler in der Prozedur C auftritt, dieser Fehler aber erst in der Fehlerbehandlungsroutine von Prozedur A berücksichtigt wird, dann wird das Programm an der durch `Resume` angegebenen Stelle in A fortgesetzt. Es ist nicht möglich, mit **Resume** aus der aktuellen Prozedur (z.B. zur fehlerhaften Anweisung in C) zu springen.

Resume Sprungmarke

- Setzt bei angegebener Sprungmarke fort.

GoTo Sprunmarke

- Setzt bei angegebener Sprungmarke fort.

On Error Resume Next

- Wenn ein Fehler auftritt, soll das Makro den Fehler ignorieren und weiter machen.

On Error GoTo Sprunmarke

- Ermöglicht die Programmierung ”echter” Fehlerbehandlungsroutinen. Anschliessend kann die Prozedur veralssen oder mit `Resume` fortgesetzt werden.
- Sprünge mit `GoTo` können das Verständnis der Funktion oder Prozedur verschleiern und entsprechen nicht den Regeln der strukturerten Programmierung. Das Kommando `GoTo` sollte daher überhaupt nicht verwendet werden. Es besteht auch keine Notwendigkeit.x
- Die Sprunmarkre (endet mit Doppelpunkt) muss innerhalb der aktuellen Prozedur liegen. Die Sprungmarke steht am Ende der Prozudur und vor der Sprungmarke steht ***Ex***`it Sub / Exit Function.`
- Innerhalb der Fehlerbehandlungsrouting können Sie mit `Err()` die Nummer des aufgetretenen Fehlers feststellen und dementsprechend darauf reagieren. Eine Liste aller in VBA vorgesehenen **Fehlernummern finden Sie in der Online-Hilfe unter dem Thema ”Fehlernummern”.** Zur Reaktion auf den Fehler ist es durchaus möglich, andere Prozeduren aufzurufen (z.B. zum Speichern von Daten).
- Beachten Sie aber, dass sowohl innerhalb der Fehlerbehandlungsroutine als auch in den von ihr aufgerufenen Prozedur weitere Fehler auftreten können. `On Error GoTo Sprungmarke` gilt nur für *einen* Fehler. Der nächste Fehler bewirkt wieder das normale Verhalten (also die Anzeige des dialogs ”Makrofehler”). Sie können das vermeiden, indem Sie innerhalb der Fehlerbehandlungsroutine abermals `On Error GoTo xxx` On oder besser – so wie hiers auf jeden Fall beschrieben ist – mit `Error GoTo xxx` ausführen.
- Verlassen Sie sich in Ihrer Fehlerbehandlungsroutine nicht darauf, dass es gelingt, den Fehler tatsächlich zu beheben. Berücksichtigen Sie auch die Möglichkeit, dass in der Fehlebehandlungsroutine selbst ein Fehler auftritt! Vermeiden sie unbedingt eine daraus resultierende Endlosschleife (ein Fehler führt zum Aufruf der Fehlerbehandlungsroutine, von dort wird die Prozedur fortgesetzt, der Fehler tritt abermals auf, neuer Aufruf der Fehlerbehandlungsroutine etc.)!

On Error GoTo 0

- Um eine zuvor eingerichtete Fehlerbehandlungsroutine zu deaktiveren. Nach `On Error GoTo 0` gilt wieder das ”normale” Verhalten von VBA, also die Anzeige des Makrofehler-Dialogs

Fehlerursache ermitteln

- Wenn ein Fehler auftritt, werden die Eigenschaften des *Err*-Objekts mit Informationen gefüllt, die den Fehler sowie die Informationen, die zur Verarbeitung des Fehlers verwendet werden können, eindeutig kennzeichnen. Jeder Fehler hat in Excel eine eindeutige Nummer, die Sie abfangen können.

Die Funktionen Err

- **Err** liefert eine Identifizierungsnummer des aufgetretenen Fehlers. Die möglichen Fehlernummern sind in der Online-Hilfe angegeben. `Error()` liefert den Fehlertext des zuletzt aufgetretenen Fehlers. `Error(n)` liefert den Fehlertext zur Fehlernummer `n`.

Die Funktionen Error

- Mit dem Kommando `Error` kann ein Fehler simuliert werden. Das ist z.B. zum Test der Fehlerbehandlunsgsroutine sinnvoll. Die Anweisung `Error n` führt zur Anzeige des Dialogs "Makrofehler" und kann daher auch am Ende einer Fehlerbehandlungsroutine sinnvoll sein, wenn es nicht gelungen ist, den Fehler zu beseitigen.
- Error übergibt als Funktionswert die Fehlermeldung im Klartext, die dem übergebenen Fehlercode zugeordnet ist. Entsprechend informiert in einer Fehlerroutine der Ausdruck: `MsgBox (Error(Err.Number))` im Klartext über den aufgetretenen Fehler.

Die Funktionen CVErr

- Mit der Funktion `CVErr` kann ein Fehlerwert für eine `Variant`-Variable erzeugt werden. Die Funktion kann z.B. dazu eingesetzt werden, bei einer benutzerdefinierten Tabellenfunktion statt eines Ergebnisses einen Fehlerwert zurückgeben:
- Result = `CVErr(xlErrValue)`
- Eine Liste der für diesen Zweck vordefinierten Fehlerkonstanten fonden Sie in der **Online-Hilfe zum Tema "Fehlerwerte in Zellen".**

Die Funktion IsError

- Eine Weitere Möglichkeit, Fehler zu umgehen, bietet die Funktion *IsError*. Diese Funktion liefert den Wert *True* zurück, wenn der Ausdruck fehlerhaft ist. Wenden Sie diese Funktion an, indem Sie innerhalb eines markierten Bereichs alle Zellen, die Fehlerwerte liefern, mit der Zahl 0 überschreiben.

```
Sub FehlerzellenAufNull()
Dim Zelle As Range
For Each Zelle In Selection
  If IsError(Zelle.Value) Then
    Zelle.Value = 0
  End If
Next Zelle
```

Diverses

- Variant-Variable für Fehlernummern.
- Resume = Weiter
- Error.Err =
- `Do Until Err.Number = 0`
- Fehlersuche engl. "Debugging"

Fehler löschen

- `Err.Number = 0`
- oder `Err.Clear`
- VBA vergisst, dass ein Fehler aufgetreten ist.

Beispiel eines Meldungsdialogs, wenn ein Fehler aufgetreten ist

- `MsgBox ”Bitte benachrichtigen Sie den Programm-Entwickler über den Fehler: ” & ”Fehlernummer: ” & Err.Number & vbCrLf & vbCrLf & Err.Description`

- `vbCrLf` = Carriage Return Line Feed (=Wagenrücklauf bzw. Zeilenumbruch) Wie `Chr(13)`

Select Case-Verzweigung, die die Fehlernummer ermittelt und entsprechend handelt.

- Anstatt `If Err.Number <> 0` einfach `If Err.Number` schreiben

```
If Err.Number Then
  Select Case Err.Number
    Case 11
      MsgBox(”Sie versuchen, durch 0 zu dievidieren!”)
    Case 6
      MsgBox(”Sie versuchen, durch 0 zu dievidieren!”)
    Case Else
    MsgBox ”Fehler ” & Err.Number & ”: ” & Err.Description, _
      vbCritical, ”prozedurname”
  End Select
Else
  MsgBox(”Alles Okay!”)
End If
```

Verzweigung zu einer Fehlerbehandlungsroutine

```
Sub DateiÖffnen
On Error GoTo Fehler
  Workbooks.Open Filename:=(”Q:\Daten\Mappe1.xls), Notify:=False
  Exit Sub
Fehler:
  (”Bitte Warten Sie bis die Datei wieder frei ist.”)
End Sub
```

Beispiel zur Fehlersimulation durch Err.Raise

- Methode Raise auf `Err.Objekt` anwenden
- `Err.Raise Fehlercode`

```
Public Sub Fehler4()
On Error GoTo DiskNotReady
  Err.Raise 71
  Exit Sub
DiskNotReady:
  If Err.Number = 71 Then
    MsgBox(”Keine Disk eingelegt oder Klappe geöffnet”)
  End If
  Resume Next
End Sub
```

Beispiel zur Fehlerdefinition mit Err.Raise

- `Err.Raise` können Sie ausser zur Fehlersimulation auch zur Fehlerdefinition verwenden.

```
Public Sub Fehler5()
Dim Plz
On Error GoTo PlzFalsch
Eingabe:
  Plz = InputBox(”Pl:”)
  If Plz < 10000 Or Plz > 99999 Then Err.Raise (vbObjectError + 1)
  'Anweisungen
Exit Sub

PlzFalsch:
  If Err.Number = (vbObjectError + 1) Then
    MsgBox (”Falsche Plz”)
    Resume Eingabe
  Else
    MsgBox (”Unbekannter Fehler”)
    Resume Next
  End If
End Sub
```

Syntax der Err.Raise-Methode

`Err.Raise(number, source, description, helpfile, helpcontext)`

- Beschreibung siehe Online-Hilfe

vbObjectError

- Die vordefinierte Excel-VBA-Konstante `vbObjectError` enthält den höchsten von VBA selbst verwendeten Fehlercode. Benutzen Sie den Ausdruck `vbObjectError + 1` für Ihren ersten eigendefinierten Fehlercode.

Fehlerbehandlung in verschachtelten Prozeduren

- Angenommen, durch das Anklicken eines Symbols wird das Unterprogramm A aufgerufen, a ruft B auf und B ruft C auf. Wenn nun in C ein Fehler auftritt, wird die zu C gehörige Fehlerbehandlungsroutine aufgerufen. Existiert in C kenie Fehlerbehandlungsroutine, dann erfolgt ein Rücksprung in B, existiert auch dort keine, wird das Programm mit der Fehlerbehandlungsroutine von A fortgesetzt. Nur wenn auch in A keine Fehlerbehandlungsroutine von vorgesehen ist, erscheint der bekannte Makrofehler-Dialog.
- VB durchsucht also die sich gegenseitig aufrufenden Unterprogramme bzw. Funktionen in umgekehrter Reihenfolge nach einer geeigneten Fehlerbehandlungsroutine. Nur wenn auch in der ursächichen Ereignisprozedur keine geeignete Fehlerbehandlungsroutine gefunden wird, meldet sich VB mit einer Fehlermeldung und bricht das Programm ab.

Reationen auf Programmunterbrechung

- Wenn Sie vermeiden möchten, dass der Anwender Ihr Programm einfach mit CTRL+Untbr stoppen kann, bestehen zwei Möglichkeiten:

- Durch `Application.`**`EnableCancelKey = xlDisabled`** erreichen Sie, dass auf das Drücken von CTRL+Untbr überhaupt keine Reaktion erfolgt. Der Vorteil dieser Massnahme besteht darin, dass dazu nur eine einzige Anweisung (in der `Auto_Open`-Prozedur) erforderlich ist.

- Wenn Sie **`EnableCancelKey`** dagegen die Konstante **`xlErrorHandler`** zuweisen, dann tritt jedesmal, wenn der Anwender CTRL+Untbr drückt, ein Fehler mit der Fehlernummer 18 auf. Sie können diesen ”Fehler” ganz normal wie andere Fehler in einer Fehlerbehand-

lungsroutine abfangen. Der Nachteil ist offensichtlich: Es muss jede Prozedur mit einer Fehlerbehandlungsroutine ausgestattet werden. Eine andere Variante besteht darin, Unterbrechungen nur in solchen Programmteilen zuzulassen, in denen sehr zeitaufwendige Berechnungen durchgeführt werden.

- Die normale Reaktion auf Unterbrechungen, also das Anzeigen einer Fehlermeldung, können Sie durch die Zuweisung **`EnableCancelKey = xlInterrupt`** wiederherstellen.

Fehler vor dem Progarmmstart melden

- Einige Tippfehler erkennt VBA, manche Fehler können erst Kompiliert festgestellt werden, manche sogar erst bei der Ausführung des Codes. In der Defaulteinstellung werden nur die Programmteile kompiliert, die tatsächlich benötigt werden. Daher kann es vorkommen, dass formale Fehler erst nach einiger Zeit, d.h., wenn die jeweilige Prozedur zum ersten Mal benötigt wird, entdeckt wird.
- Dazu können Sie entweder das gesamte Projekt mit Debuggen/Kompilieren in Pseudo-Code umwandeln oder die beiden Kompiliereinstellungen in ”Extras/Optionen” deaktivieren.

Fehler bei der Variablendeklaration

- *Ohne* die Option `Explicit` interpretiert VBA ein falsch geschriebenes Schlüsselwort in der Regel wie eine nicht deklarierte Variant-Variable. Verwenden Sie daher immer die `Option Explicit`!
- In Extras / Optionen / KK: Bei jedem Fehler bedeutet, das jeder Fehler selbst dann zu einer Programmunterbrechung führt, wenn dieser Fehler durch `On Error` abgefangen würde.

Programmänderung im laufenden Programm

- Sie können währen das Programm lauft, diverse Änderungen am Programm vornehmen und anschliessend das Programm fortsetzen. Das ist zum Beseitigen von Fehlern natürlich äusserst praktisch. Eine Fortsetzung ist allerdings nicht möglich, wenn sich die Struktur des Programms ändert, also etwa die deklaration der Parameter einer gerade aktiven Prozedur. Fall Sie die Option Extras / Optionen / Allgemein / KK: Benachrichtigung vor Zustandsänderung aktivert haben, warnt Sie die Entwicklungsumgebung vor solchen Änderungen.Meistens ist es angenehmer, vor dem Progarmmstart *alle* formalen Fehler aufzusprüren.

15.1 Fehlernummern

- Eine komplette Liste an auffangbaren Fehlern können Sie der Online-Hilfe entnehmen, wenn sie im Index den Suchbegriff Auffangbarer Fehler eingeben.
- Klicken Sie auf den Hyperlink der Fehlerbezeichnung, um weitere Informationen zum Fehler und eine Beschreibung zu bekommen, wie Sie diesen Fehler vermeiden können.

429

```
Sub WordMitNeuemDokumentStarten()
Dim WordObj As Object
Dim WordDoc As Object
  On Error Resume Next
  Set WordObj = GetObject(, ”word.application.9”)
  If Err.Number 429 Then
    Set WordObj = CreateObject(”word.application.9”)
    Err.Number = 0
```

```
  End If
End Sub
```

Haben Sie Word bereits gestartet, kommt es zu keinem Fehlerfall beim Versuch, Word über die Funktion `GetObject` zu aktivieren. Im Fall dessen, dass Word noch nicht geöffnet ist, meldet Excel die Fehlernummer `429`**,** welche sinngemäss besagt, dass es nicht möglich war, die Applikation zu aktivieren. In diesem Fall können Sie die Anweisung If einsetzen, um gezielt darauf zu reagieren. Ihre Raktion besteht nun darin, über die Funktion

15.2 Syntaxzusammenfassung

Fehlersuche

Debug.Print …	Ausgabe im Testfenster
MsgBox	Ausgabe im Meldungsfenster
Stop	Programm unterbrechen

Reaktion auf Programmfehler

On Error Resume Next	nächste Anweisung ausführen
On Error GoTo label	Fehlerbehandlungsroutine aufrufen
On Error GoTo 0	normale Reaktion: Makro-Dialog

Kommandos und Funktionen in der Fehlerbehandlungsroutine

Resume	führt fehlerhafte Anweisung neuerlich aus
Resume Next	setzt Prozedur mit nächster Anweisung fort
Resume label	setzt Prozedur bei der Sprungmarke fort
Err	ermittelt die aktuelle Fehlernummer
Error(n)	ermittelt den Text zur Fehlernummer
Error n	simuliert einen Fehler
CVE(n)	verwandelt n in einen Fehlerwert (zur Rückgabe)

Reaktion auf Programmunterbrechung

Application.EnableCancelKey	bestimmt die Raktion auf Strg+Untbr. Erlaubt Werte: xlInterrupt, xlDisabled, xlErrorHandler

16 Dialoge

16.1 Die MsgBox

Das Meldungsfenster (MessageBox)

Syntax der MsgBox-Funktion

```
MsgBox(prompt[, buttons] [, title] [, helpflile, context]
MsgBox ”Meldung”, Schaltflächen, ”Titel”

Rüchgabewert = MsgBox(Meldung [, Schaltflächen] [, Titel])
Rückgabewert = MsgBox(”Meldung”, Schaltflächen, ”Titel”)

MsgBox(”Zeichenkette ” + Chr(13) + ”…”)
MsgBox(”Text”, 289, ”Titel)
```

Die Schlatflächen für MsgBox

- Konstante oder Wert		
vBOKOnly	0	zeigt nur die Schaltfläche OK an
vbOKCancel	1	zeigt die Schaltflächen OK und Abbrechen an
vbAbortRetryIgnore	2	zeigt die Schaltflächen Abbruch, Wiederholen und Ignorieren an
vbYesNoCancel	2	zeigt die Schaltflächen Ja, Nein und Abbrechen an.
vbYesNo	4	zeigt die Schaltflächen Ja und Nein an
vbRetryCancel	5	zeigt die Schaltfläche Wiederholen und Abbrechen an.
vbDefaultButton1	0	Erste Schaltfläche ist Standardfläche
vbDefaultButton2	256	Zweite Schaltfläche ist Standardfläche
vbDefaultButton3	512	Dritte Schaltfläche ist Standardfläche
vbDefaultButton4	768	Vierte Schaltfläche ist Standardfläche
vbApplicationModal	0	Anwendungsgebunden (Wechsel zu anderen Anwendungen möglich)
vbSystemModal	4096	Systemgebunden (Kein Wechsel zu anderen Anwendungen)
VbMsgBoxSetForeground	65536	Setzt die MsgBox in den Vordergrund
vbMsgBoxHelpButton	16384	fügt dem Meldungsfenster eine Hilfeschaltfläche hinzu

Ausrichtung

	0	Linksbündig
vbMsgBoxRight	524288	Rechtsbündig
vbMsgBoxRtlReading	1048576	Text wird von rechts nach links geschrieben

Symbole

vbCritical	16	zeigt Melgung mit Stop-Symbol an.
vbQuestion	32	zeigt Melgung mit Fragezeichen-Symbol an.

vbExclamation	48	zeigt Melgung mit Ausrufezeichen-Sybmol an.
vbInformation	64	zeigt Meldung mit Info-Symbol an.

Welche Schaltfläche wurde geklickt

vbOK	1	Die Schaltfläche OK wurde geklickt
vbCance	2	Die Schaltfläche Abbrechen wurde geklickt
vbAbort	3	Die Schaltfläche Abbruch wurde geklickt
vbRetry	4	Die Schaltfläche Wiederholen wurde geklickt
vbIgnore	5	Die Schaltfläche Ignore wurde geklickt
vbYes	6	Die Schaltfläche Ja wurde geklickt
vbNo	7	Die Schaltfläche Nein wurde geklickt

Löschrückfrage einholen

```
Sub LöschenMarkierterBereich()
Dim i As Integer
i = MsgBox(”Wollen Sie den markierten Bereich wirklich löschen?”, _
          1 + vbQuestion, ”Löschenabfrage”)
If i = 2 Then Exit Sub
Selection.Clear
End Sub
```

16.2 Imputbox

Das Eingabefenster (InputBox)

- InputBox gibt immer eine Zeichenkette zurück. Mit Val-Funktion in eine echte Zahl umwandeln.

Syntax der Imputbox-Funktion

```
Rückgabewert _
  = InputBox(”Eingabeaufforderung”, ”Titel”, ”Schaltflächen”)

InputBox (Einabeaufforderung ,Titel, Default, XPosition, Yposition,
Helpfile, HelpContext) As String

a = InputBox(Default:=”117”, Title:=”bildermakro”, _
  Promt:=” Wie viele Bilder enthält der Text?”)
```

16.3 Application.Inputbox

- InputBox wird von der VBA-Bibliothek zur Verfügung gestellt. Application.InputBox wird von der Excel-Bibliothek zur Verfügung gestellt.
- Entscheidend für die Anwendung von `Application.InputBox` ist der letzte Parameter, mit dem der Typ der Eingabe angegeben werden kann, sofern keine Texteingabe durchgeführt werden soll.

Syntax der Application.Imputbox-Funktion

```
Rückgabewert = Application.InputBox(promt, title, default, Left, _
Top, helpFile, helpContext, type)
```

Type

- Legt den Datentyp des Rückgabewertes fest. Ohne Angabe dieses Arguments gibt das Dialogfeld den Daentyp Text zurück. Entnehmen Sie die zulässigen Rückgabewerte der nächsten Tabelle.

Werte

0	Formel
1	Zahl
2	Text
4	logischer Wert (True oder False)
8	Zellenbezug
16	Fehlerwert
64	Wertematrix

- Bei Typ 64 gibt `InputBox` ein `Variant`-Feld mit den Werten des angegebenen Zellbereichs zurück.

Beispiel zu Type 8

```
Dim b As Range
Set b = Application.InputBox(”Geben Sie einen Zellbereich an!”, Type:=8)
```

- `InputBox` liefert als Ergebnis die getroffene Auswahl in *dem* durch den Type-Parameter definierten Format. Falls der Anwender die Eingabe mit Abbrechen beendet, liefert die Methode den Wahrheitswert `False`. Dieser Umstand macht die Auswertung der Eingabe für `Type:= 8` nicht gerade einfach: Da das Ergebnis normalerweise ein `Range`-Objekt ist, muss die Zuweisung des Ergebnisses durch *Set* erfolgen. Dabei komme es aber zu einem Fehler, wenn `InputBox` statt eines `Range`-Objekts nur den Wert `False` zurückgibt. Daher sieht ein ordnungsgemässer Code so aus:

```
Dim b As Range
On Error Resume Next
Set b = Application.InputBox(”Geben Sie einen Zellbereich an!”, Type:=8)
If Err <> 0 Then ’da ist ein Fehler aufgetreten
  MsgBox ”Das war wohl ken Zellbereich!?”
End If
```

16.4 Integrierte Dialoge

Vordefinierte Excel-Standarddialoge

- Alle Excel Dialoge können über die `Application- Dialogs` Methode ausgewählt und und mit `Show` angezeigt werden
- Die Ausführung der `Show`-Methode ist nur dann möglich, wenn gerade ein geeignetes Objekt aktiv ist.
- Die Methode `Show` kann sowohl als Kommando als auch als Funktion verwendet werden. Im zweiten Fall gibt sie `True` zurück, wenn der Dialog ordnungsgemäss mit OK beendet wurde, oder `False`, wenn der Dialog mit Abbrechen, Esc oder über das Fensterschliessfeld abgebrochen wurde

```
Ergebnis = Application.Dialogs(xlDialogArrangeAll).Show
```

Übergabe von Parametern an einen Dialog

- Eine Liste aller `xlDialog`-Konstanten finden Sie im Objektkatalog (Bibliothek: Excel, Objekt: `xlBuiltinDialogs`). An die Methode `Show` können bis zu 30 Parameter übergeben werden, mit denen Voreinstellungen im Dialog gewählt werden. Eine sehr knappe Beschreibung der Parameter finden Sie in der Online-Hilfe unter dem Link "Integrierte Dialog-Argumentenliste" im `Show`-Hilfetext. Aber auch damit bleibt die Übergabe von Parametern ein mühsames und unübersichtliches Unterfangen. Das wird an einem Beispiel deutlich: Der Dialog zum Öffnen einer Excel-Datei `(xlDialogOpen)` ist in der Online-Hilfe folgendermassen beschrieben:

```
xlDialogOpen file_text, update_link, read_only, format, prot_pwd,
write_res_pwd, ignore_rorec, file_origin, custom_delimit,
add_logical, editable, file_access, notify_logical, converter
```

- `Show` verwendet wie alle anderen VBA-Methoden den Mechanismus der benannten Parameter. Da `Show` aber für sehr viele unterschiedliche Dialoge herhalten muss, sind die Parametername recht simpel ausgefallen: `Arg1, Arg2, Arg3` etc. Wenn Sie beim Öffnen einer Datei das Optionsfeld `Schreibgeschützt` aktivieren möchten, sieht die entsprechende Anweisung folgendermassen aus:

```
Ergebnis = Application.Dialogs(xlDialogOpen). Show(Arg3:=True)
```

Dialog öffnen aufrufen

```
Sub DialogÖffnen()
Dim b As Boolean
  b = Application.Dialogs(xlDialogFindFile).Show
  MsgBox b
End Sub
```

Der Dialog Öffnen mit automatischer Passworteingabe

```
Sub DialogÖffnenMitPassword()
Application.Dialogs(xlDialogOpen).Show _
  "C:\eigene Dateien\Mappe1.xls", arg5:="test"
End Sub
```

Der Dialog Drucken aufrufen

```
Sub DialogDrucken()
Const SeiteVon = 1
Const SeiteBis = 4
Const Kopien = 2
    Application.Dialogs(xlDialogPrint).Show arg1:=2, _
    arg2:=SeiteVon, arg3:=SeiteBis, arg4:=Kopien
End Sub
```

Den Dialog Blattschutz anzeigen

```
Sub DialogBlattschutz()
 Application.Dialogs(xlDialogProtectDocument).Show
End Sub
```

Den Dialog Optionen aufrufen

```
Sub DialogExtrasOptionenBearbeiten()
 Application.Dialogs(xlDialogOptionsEdit).Show
End Sub
```

Dialoge zur Dateiauswahl

```
xlDialogOpen
xlDialogSaveAs
```

oder mit

```
GetOpenFilename
GetSaveAsFilename
```

- Damit werden ebenfalls die Dateiauswahldialoge ausgewählt, allderdings wird lediglich der ausgewählte Dateiname zurückgegeben (ohne eine Datei zu laden oder zu speichern). Sie sind mit diesen Methoden also flexibel, was die weitere Reaktion anbelangt

```
Dateiname = Application.GetSaveAsFilename
```

Tastatureingaben in Dialogen simulieren mit SendKeys

- `Show` zeigt zwar den Dialog an, die Eingabe von Parametern bleibt aber weiter dem Anwender Ihres Programms überlassen. Manchmal kann es sinnvoll sein, auch eine Tastatureingabe zu simulieren. Dazu steth Ihnen die `Application`-Methode `SendKeys` zur Verfügung. Wesentlich bei der Anwendung dieser Methode ist der Umstand, dass sie *vor* der Anzeige des Dialogs ausgeführt werden muss, was eigentlich unlogisch erschient. Begründung: Windows speichert die sumilierte Tastenfolge in einem Tastaturpuffer und führt die Tastatur erst dann aus, wenn es dazu Gelegenheit findet etwa nach der Anzeige eines Dialogs.

- Das folgende Fenster zeigt den Dialog zur Anordnung der Fenster an, wählt darin aber mit Alt + O gleich die Option Horizontal aus. Der Anwender braucht den Dialog nur noch durch Return zu bestätigen.

```
SendKeys "%O"
Application.Dialogs(DialogArrangeAll).Show
```

- Die Syntax der Zeichenkette, in der die simulierte Tastatureingabe in `SendKeys` angegeben wird, ist ausführlich in der Online-Hilfe zu dieser Methode beschrieben. Prinzipiel ist es mit `SendKeys` auch möglich, die Eingabe im Dialog durch OK (also durch die Sumulation von Return) abzuschliessen. Auf diese Weise können Sie diverse Excel-Kommandos über den Umweg eines Dialogs direkt ausführen. Wenn die Eigenschaft `ScreenUpdating` auf `False` gestellt ist, sieht der Anwender Ihres Programms den Dialog nicht einmal. Dennoch ist diese Vorgehensweise nicht zu empfehlen (langsamer, nicht International gültig, in neuer Version wird das Programm nicht mehr laufen)

Die Datenmaske *ShowDataForm*

Der Dialog zur Auswahl, Veränderung und Neueingabe von Datensätzen wird nicht mit `Dialogs(…).Show`, sondern mit `ShowDataForm` aufgerufen.

Sihe auch Datenmaske aus Seite 207

17 UserForms (MS-Forms-Bibliothek)

17.1 Diverses

Hide und Unload

- Wenn Sie den Dialog mit `Unload Me` statt mit `Hide` schliessen, wird damit nicht nur der Dialog aus dem Speicher entfernt, es gehen auch alle im dazugehörigen Modul definierten Variablen verloren.

vbModeless

- Mit `Show VbModeless` erreichen Sie, dass der Dialog nicht modal angezeigt wird. Der Anwender kann also in Excel weiterarbeiten, ohne den Dialog vorher verlassen zu müssen. `VbModeless` sollt nicht gleichzeitig mit dem `RefEdit`-Steuerelement eingesetzt werden.

18 Die MS-Forms-Steuerelemente

18.1 Diverses

Zugriffstaste festlegen

`Accelerator`-Eigenschaft

Aktiverreihenfolge

- Die `TabIndex`-Eigenschaft verändern. Das erste Steuerelement hat den Wert **0**.
- Falls Sie einzlene Steuerelemente von der Aktivierreihenfolge ausnehmen möchten, setzen Sie einfach deren `TabStop`-Eigenschaft auf `False`.
- Der Eingabefokus kann auch im Programmcode durch die Methode ***SetFocus*** verändert werden.

```
Dialogname.Steuerelementname.SetFocus
```

Zusätzliche Steuerelemente (ActiveX-Steuerelemente)

- Im Werkzeugfenster werden fünfzehn MS-Forms-Steuerelemente angezeigt. Falls Sie mit Office Developer arbeiten oder neben Office auch andere Programme installiert haben, stehen Ihnen eine ganze Menge weiterer Steuerelemente zur Verfügung. In MS-Forms-Dialogen können nämlich alle ActiveX-Steuerelemente verwendet werden, die auf Ihrem Rechner installiert sind.

Kontextmenü: Zusätzliche Steuerelemente

- Wohl wird die entsprechende Objektbibliothek bei der Auswahl eines Steuerelements automatisch aktiviert, wenn Sie das Steuerelement wieder entfernen, bleibt der Bibliothekverweis allerdings bestehen. Sie sollten den Verweis explizit in Extras / Verweise wieder deaktiveren.

- Wenn Sie Excel-Anwendungen weitergeben möchten, achten Sie darauf, dass Sie nur solche Steuerelemente verwenden, von denen Sie wissen, dass Sie auch am Rechner Ihrer Kunden stehen. Nurd wenn Sie mit Office Developer arbeiten, können Sie Zusatzsteuerelemente im Rahmen eines Setup-Programms weitergeben (aber natürlich auch nur, soweit dies lizenzrechtlich zulässig ist).

Neue Steuerelemente erstellen

- Sie können einzelne oder mehrere (als Gruppe) Steuerelemente vom Dialog in das Werkzeugfenster verschieben. Z.B. Ok und Abbrechen. Der Programmcode wird nicht aufgenommen.
- Falls Sie VB 6 besitzen und ausreichend Programmiererfahrung haben, könen Sie selbst *wirklich* neue Steuerelemente, also unabhängig von den Grenzen, die durch MS-Forms-Bibliothek gegeben sind.

18.2 Gemeinsames

Gemeinsame Merkmale

- `Cancel` und `Default` für ”Abbruch”- und ”OK”-Button
- Mit `ControllSource` können Sie den Inhalt eines Steuerelements mit dem Inhalt einer Zelle verbinden. Änderungen in Tabelle spiegelt sich automatisch im Inhalt des Steuerelements wieder. (Bei Listenfeldern kann mit `RowSource` die ganze Liste mit einem Zellbereich einer Tabelle syncronisieren.
- `Tag` hilft bei der Verwaltung von Steuerelementen. Zeichenkette wird nicht angezeigt.
- `Visible`-Eigenschaft steuert die Sichtbarkeit der Steuerelemente

Gemeinsame Eigenschaften

`Cancel`	`True`, wenn das Steuerelement durch `Esc` ausgewählt weden kann
`ControlTipText`	gelber Infotext (Tooltip-Text)
`ControlSource`	stellt die Verbindung zu einer Zelle aus einem Tabellenblatt her
`Default`	`True`, wenn das Element durch Return ausgewählt werden kann
`RowSource`	stellt die Verbindung zu einem Zellbereich her (für Listenfelder)
`TabIndex`	Nummer, die die Aktivierungsreihenfolge angibt
`TabStop`	`True`, wenn das Steuerelement mit Tab ausgewählt werden kann
`Tag`	unsichtbare Zusatzinfo, die manchmal bei der Erwaltung hilf
`Visible`	`True`, wenn das Steuerelement sichtbar ist

Gemeinsame Methoden

`SetFocus`	richtet den Eingabefokus in ein Steuerelement

Gemeinsame Ereignisse

`Enter`	das Steuerelement hat den Eingabefokus erhalten
`Error`	es ist ein Fehler aufgetreten
`Exit`	das Steuerelement hat den Eingabefokus vorloren

18.3 Bezeichnungsfeld (Label)

- Der Text wird über die Eigenschaft `Caption` eingestellt und darf sich über mehrere Zeilen erstrecken. In diesem Fall muss `WordWrap` auf `True` gesetzt werden.
- Der Text kann wahlweise linksbündig, rechtsbündig oder zentriert angezeigt werden (`TextAlign`).
- Wenn `AutoSize` auf `True` gesetzt wird, passt sich die Grösse des Labels automatisch an
- Die Schriftfarbe kann mit `Font`, `Back`- und `ForeColor` frei eingestellt werden.
- Weitere Gestaltungsmöglichkeiten ergeben sich aus der Umrandung (`BorderStyle`, `BorderColor`) und der Eigenschaft `SpecialEffect`, mit der 3D-Effekte wie bei Buttons erzielt werden können.
- Schliesslich kann im Label sogar eine Bitmap angezeigt werden (`Picture`, `PicturePosition`). Sie sehen also, das das Labelfeld mehr hält, als es verspricht.

Label – Eigenschaften

`AutoSize`	Grösse des Steuerelements passt sich an den Text an
`BackColor`	Hintergrundfarbe
`BorderColor`	Randfarbe
`BorderStyle`	Umrandung ein / aus
`Caption`	der angezeigte Text
`Font`	Schriftart
`ForeColor`	Schriftfarbe
`Picture`	Bitmap
`PicturePosition`	Position, an der die Bitmap angezeigt wird
`SpecialEffect`	3D-Effekte
`TextAlign`	Textausrichtung (links, rechts, mittig)
`WordWrap`	Zeilenumbruch

Label – Ereignis

`Click`	das Labelfeld wurde angeklickt

18.4 Textfeld (TextBox)

- Anzahl Zeichen: `Len(textfeld.Text)`
- Anzahl Zeilen: Eigenschaft `LineCount`
- Aktuelle Zeile: Eigenschaft `CurLine`
- Mit den Eigenschafte `MulitLine` und `ScrollBars` wird eine mehrzeilige Texteingabe und gegebenenfalls die Anzeige von Bildlaufleisten zugelassen.
- Mit PasswortChar können Sie ein Textzeichen einstellen, üblicherweise *, das statt des eingegebenen Texts angezeigt wird.
- Mit `EnterFieldBehavior=0` erreichen Sie, dass beim Anklicken des Textfelds automatisch der gesamte Inhalt markiert wird
- `EnterKeyBehavior` steuert das Verhalten bei `Return`. Wenn die Eigenschaft auf `True` gesetzt wird, bewirkt `Return` die Eingabe einer neuen Zeile. Ist die Eigenschaft dagegen auf `False` gestellt, wählt `Return` den mit `Defautl=True` markeirten Button aus. Zur eingabe einer neuen Zeile muss dan CTRL+Return verwendet werden
- Eine analoge Bedeutung hat `TabKeyBehavior`: Die Eigenschaft gibt an, ob im Steuerelement `Tab` dem Steuerelementwechsel vorbehalten ist.
- Linker Rand des Textfelds, der die bequeme Markierung der ganzen Zeile ermöglicht, ist bei einzeiligen Textfeldern überflüssig. Desshalb Eigenschaft `SelectionMargin` auf `False` setzen
- Zugriff auf markierten Text mit `SelText`
- Eigenschaften `SelStart` und `SelLength` geben das erste Zeichen der Markierung bzw. die Länge der Marierung an.

```
With textfeld
  .SelLegth = 0                  'Markierung auflösen
  .SelStart = 0: .SelLenght = 5 'Die ersten fünf Zeichen markieren
  .SelText = ""                  'den markierten Text löschen
  .SelStart = 10                 'Eingabecursor an neue Position
  .SelText = "abc"               'Dort drei Zeichen einfügen
End With
```

- Mit den Methoden `Copy` und `Cut` können Sie den gerade markierten Text in die Zwischenablage kopieren bzw. ausschneiden. `Paste` ersetzt den zur Zeit markierten Text durch den Inhalt der Zwischenablage.

Ereignisse

- Das wichtigste Ereignis lautet *Cange*.
- Tastaturereignisse `KeyPress, KeyDown` und `KeyUp` siehe im nächsten Kapitel

TextBox – Eigenschaften

`CurLine`	aktuelle Zeilennummer
`EnterFieldBehavior`	0, wenn der gesamte Inhalt beim Aktivieren markiert wird
`EnterKeyBehavior`	`True`, wenn mit Return eine neue Zeile eingegeben werden kann
`LineCount`	Anzahl der Zeilen
`MultiLine`	`True`, wenn mehrere Textzeilen verwendet werden
`PasswortChar`	Platzhalterzeichen für den Text
`ScorllBars`	gibt an, ob bei langen Texten Bildlaufleisten angezeigt werden

SelectionMargin	True, wenn Text in Randspalten zeilenweise markiert werden kann
SelLength	Länge des markierten Texts
SetStart	Beginn des markierten Texts
SetText	markierter Text
TabKeyBehaviour	True, wenn mit Tab ein Tabulatorzeichen eingegeben warden kann
Text	Inhalt des Textfelds

TextBox – Methoden

Copy	markierten Text in die Zwischenablage kopieren
Cut	markierten Text in die Zwischenablage ausschneiden
Paste	Text aus der Zwischenablage einfügen

Textbox – Ereignisse

Change	der Inhalt des Textfelds hat sich geändert
KeyDown	eine Taste wurde gedrückt
KeyPress	Tastatureingabe
KeyUp	eine Taste wurde losgelassen

18.5 Tastaturereignisse KeyPress, KeyDown und KeyUp

- `KeyPress`: Dieses Ereignis tritt beim Drücken einer alphanummerischen Taste auf. An die Ereignisprozedur wird der ANSI-Code des eingegebenen Zeichend übergeben. Neben den alphanumerischen Zeichen werden auch die Tasten `Return`, `Esc` sowie `CTRL`-Kombinationen gemeldet. `KeyPress` tritt nicht auf, wenn der Benutzer Coursor- oder Funktionstasten, Delete, Insert etc. drückt und ist daher für eine allgemeingültige Tastaturverwaltung nicht ausreichend
- `KeyDown` Dieses Ereignis tritt beim Drücken einer beliebigen Taste auf. An die Ereignisprozedur wird der interne Tastaturcode der gedrückten Taste sowie der Zustandscode der Umschalttaste übergeben. `KeyPress` tritt nicht nur beim Drücken von Cursor- oder Funktionstasten auf, sondern auch dann, wenn nur `Shift` oder CTRL gedrückt wird!
- `KeyUp`: Dieses Ereignis ist das Gegenstück von `KeyDown` und tritt beim Loslassen der Taste auf. An die Ereignisprozedur werden die gleichen Parameter wie bei `KeyUp` übergeben.
- Wenn eine alphanummerische Taste gedrückt wird, ruft Visual Basic zuerst die `KeyDown`-Ereignisprozedur auf, dann `KeyPress` und schliesslich `KeyUp`. Wenn die Taste längere Zeit gedrückt bleibt, werden die `KeyDown`- und `KeyUp`-Ereignisprozeduren mehrfach aufgerufen (Auto-Repeat). Die drei Ereignisse treten nicht auf, wenn der Benutzer
  - mit Tab zwischen Steuerelementen wechselt,
  - mit Esc einen Button mit `Cancel=True` auswählt oder
  - mit Return einen Button mit `Default=True` auswählt

```
Private Sub TextBox1_KeyPress(ByVal KeyAscii As MSForms. _
ReturnInteger)
  If Chr$(KeyAscii) = ”,” Then KeyAscii = Asc(”.”)
End Sub
```

Alles weitere siehe Online-Hilfe

Listenfelder (ListBox) und Kombinationslistenfeld (ComboBox)

- **Normales Listenfeld** (`ListBox`): Die Liste wird in einem rechteckigem Bereich angezeigt, dessen Grösse bereits beim Formularentwurf festgelegt wird. Wenn nicht alle Elemente gleichzeitig angezeigt werden können, erscheint automatisch eine Bildlaufleiste.
- **DropDown-Listenfeld:** (`ComboBox` mit `Style=fmStyleDropDownList`): Wie oben, allerdings ist die Liste ausklappbar.
- **DropDown-Kombinationsfeld** (`ComboBox` mit `Style=fmStyleDropDownCombo`): Das ausklappbare Listenfeld ist mit einem Textfeld kombiniert, in dem auch Texte eingegeben werden können, die nicht einem vorhandenen Listeneintrag entsprechen. Dieses Listenfeld ermöglicht also eine Erweiterung der Liste durch Benutzereingaben.
- Alternative Darstellung: *ListeStyle=fmListStyleOption*. So wird jeder Listeneintrag als Optionsfeld angezeigt.

Zugriff auf Listenelemente:
Die einzelnen Einträge einer Liste werden mit der Methode *AddItem* an das Steuerelement übergeben. Der Zugriff auf die Listenelemente erfolgt über die Eigenschaft *List(n)*. *ListIndex* gibt den zuletzt ausgewählten Eintrag an (oder -1, falls keinen Eintrag aus der Liste gewählt wurde), *ListCount* gibt die Anzahl der Einträge der Liste an. Mit *RemoveItem* können einzelne Einträge wieder entfernt werden. *Clear* löscht die gesamte Liste

- Die Nummer des aktuell ausgewählte Listenelement ist über die Eigenschaft *Listindex* zugänglich. (Die Numerierung beginnt wie bei allen Eigenschaften des Listenfelds mit 0) *Value* enthält normalerweise denselben Wert wie Listindex (sofern *BoundColumn* in der Defaulteinstellung belassen wird). Die *Text*-Eigenschaft enthält den alt des ausgewählten Elements.
- Aus unerfindlichen Gründen ist im Textbereich des Kombinationslistenfelds wie im Textfeld ein Markierungsrand vorgesehen. Die Defaulteinstellung von *SelectionMargin* lautet zu allem Überfluß *True* (obwohl im Textbereich dieses Steuerelements ohnedies nur eine Zeile angezeigt werden kann). Setzen Sie die Eigenschaft auf *False*, um den irritierenden Rand zu beseitigen.

Mehrfachauswahl
- In normalen Listenfeldern können mehrere Einträge gleichzeitig ausgewählt werden, wenn die Eigenschaft *MultiSelect* auf *fmMultiSelectMulti (1)* oder *fmMultiSelectExtended(2)* gesetzt wird. Zur Auswertung müssen Sie in einer Schleife alle *Selected(i)*-Eigenschaft abfragen, um festzustellen, welche Listeneinträge ausgewählt wurden (Die Mehrfachauswahl erfolgt durch gleichzeitiges Drücken der Maustaste mit Shift oder Strg.)

Mehrspaltige Listenfelder
- In Listenfeldern können auch mehrere Spalten gleichzeitig angezeigt werden. Dazu muß *ColumnCount* auf einen Wert größer 1 gesetzt werden. Der Zugriff auf die einzelnen Listeneinträge erfolgt mit *List(zeile, spalte)*, wobei die Numerierung jeweils mit 0 beginnt. Zuweisungen an *List* können auch direkt durch ein zweidimensionales Feld erfolgen, also etwa *List=feld()*. In umgekehrter Richtung ist das allerdings nicht möglich.
- Falls *ColumnHead* auf *True* gesetzt wird, wird Platz für eine zusätzliche Überschriftzeile gelassen. Ein direkter Zugriff auf deren Einträge scheint unmöglich zu sein. Die Überschriften werden aber automatisch aus einer Excel-Tabelle gelesen, wenn über *RowSource* eine Verbindung zu einem Zellbereich hergestellt wird. Z.B. *RowSouce="Tabelle2!B2:D6"*. Die Überschriftenzellen aus B1:D1 liest d*s Listenfeld

selbständig. Mit *ControlSource* kann ein zusätzliches Tabellenfeld angegeben werden, das die Nummer der aktuellen Spalte enthält.

- Die Breite der Spalten wird durch *ColumnWidths* gesteuert. In der Defaulteinstellung -1 sind alle Spalten gleich breit (aber mindestens 95 Punkt; wenn das Listenfeld dafür zu schmal ist, wird eine horizontale Bildlaufleiste eingestellt). Durch die Einstellung "2cm;3cm" erreichen Sie, daß die erste Spalte 2 cm und die zweite Spalte 3 cm breit ist. Die Breite der dritten Spalte ergibt sich aus dem verbleibenden Platz.

List- und ComboBox – Eigenschaften

`BoundColumn`	Spalte, deren Inhalt in `Value` angegeben wird
`ColumnHead`	Überschriftenzeile für mehrspaltige Listen
`ColumnWidths`	Breite der Spalten
`ControlSource`	Tabellenzelle mit Nummer des ausgewählten Element
`List(n)`	Zugriff auf Listenfeldelement
`List(zeile, spalte)`	Zugriff bei mehrspaltigen Listen. Numerierung beginnt bei 0
`ListCount`	Anzahl der Listenelemente bzw. Zeilen
`ListIndex`	Nummer des ausgewählten Elements (beginnend mit 0)
`ListStyle`	Listeneinträge als Optionsfelder darstellen
`MultiSelect`	Mehrfachauswahl zulassen
`RowSource`	Tabellenbereich mit Listeninhalt (z.B. ”Tabelle1!A1:A3”)
`Style`	`fmStyleDown` oder `fmStyleDropDownColumn` (nur `ComboBox`)
`Text`	Text des ausgewählten Elements
`TextColumn`	Spalte, deren Inhalt in `Text` angegeben wird
`Value`	Nummer oder Text des Listenelements (bei `Boundcolumn >0`)

List- und ComboBox – Methode

`AddItem`	Liste erweitern
`Clear`	Liste löschen
`RemoveItem`	Listeneintrag löschen

List- und ComboBox – Ereignisse

`Change`	Elementauswahl oder Textauswahl bei `ComboBox`
`Click`	Elementauswahl
`DblClick`	Doppelklick auf ein Listenelement
`DropButtomClick`	die Dropdown-Liste soll angezeigt werden (nur `ComboBox`)

18.6 Kontrollkästchen (CheckBox) und Optionsfelder (OptionButton)

- Der aktuelle Zustand der beiden Steuerelemente wird der *Value*-Eigenschaft entnommen. Zulässige Werte sind *True*, *False* und *Null*. (Die Einstellung *Null* markiert einen Undefinierten Zustand.
- Aus unerfindlichen Gründen ist es unmöglich, die *Value*-Eigenschaft im Dialog-Editor voreinzustellen. Sie müssen statt dessen entsprechende Anweisungen in *User_Initialize* ausführen.

CheckBox, OptionButton – Eigenschaften

`Caption`	Beschriftungstext
`TripleState`	auch ”undefiniert” (*Null*) als Eingabe zulassen
`Value`	aktueller Zustand

CheckBox, OptionButton – Ereignis

Click	der Zustand hat sich geändert

18.7 Buttons (CommandButton) und Umschaltbutton (ToggleButton)

CommandButton, ToggleButton – Eigenschaften

AutoSize	Buttongrösse an Inhalt (Text/Grafik) anpassen
Cancel	Auswahl durch Esc
Caption	Beschriftungstext
ControlTipText	gelber Infotext
Default	Auswahl durch Return
Picture	Grafik
PicturePosition	Position der Grafik
TakeFocusOnClick	die Einstellung *False* verhindert, dass der Button beim Anklicken den Eingabefokus erhält (wichtig in Tabellenblätter)
TripleState	auch ”undefiniert” (Null) als Eingabe zulassen
Value	aktueller Zustand

CommandButton, ToggleButton – Ereignis

Click	der Button wurde angeklickt

18.8 Rahmenfeld (Frame)

- Der Inhalt eines Rahmenfelds kann mit Bildlaufleisten ausgestattet werden. Zur Anzeige von Bildlaufleisten muß *ScrollBars* auf *fmScrollBarsBoth* gesetzt werden. Wenn die Bildlaufleisten automatisch verschwinden sollen, sobald der gesamte Inhalt des Rahmenfelds sichtbar ist, empfiehlt sich außerdem die Einstellung *KeepScrollBarsVisible =fmScrollBarsNone*.
- Damit das Rahmenfeld weiß, wie groß der verschiebbare Inhalt ist, müssen außerdem die Eigenschaften *ScrollWidth* und *ScrollHeight* mit Werten belegt werden. Die passenden Einstellungen müssen zumeist im Programmcode (etwa in *Form_Load*) ermitttelt werden. Die Kommandos in *Form_Initiulize* bewirken, daß der verschiebbare Bereich der gerade sichtbaren Innengröße des Rahmenfelds entspricht. (Bildlaufleisten werden damit erst dann erforderlich, wenn entweder der Zoom-Faktor vergrößert ode Rahmenfeld verkleinert wird.) *InsideWidth* und *-Height* geben den nutzbaren Innenbereich des Rahmenfelds an.
- Über die *Controls*-Aufzählung kann auf alle Steuerelemente innerhalb des Rahmens zugegriffen werden. *ActiveControl* verweist auf das aktive Steuerelement innerhalb des Rahmens. Die Methoden *AddControl* und *RemoveControl* ermöglichen ein bequemes einfügen und Entfernen von Steuerelementen.

Frame – Eigenschaften

`ActiveControl`	actives Steuerelement innerhalb der Gruppe
`Controls`	Zugriff auf die enthaltenen Steuerelement
`InsideWidth/-Height`	Grösse des nutzbaren Innenbereichs
`KeepScrollBarsVisible`	Bildlaufleiste immer anzeigen
`ScrollBars`	gibt an, ob Bildlaufleiste verwendet warden soll
`ScrollLeft/-Top`	linke obere Ecke des sichtbaren Bereichs
`ScrollWidth/-Height`	Grösse des verschiebbaren Bereichs
`Zoom`	Vergrösserungsfaktor für Inhalt des Rahmens

Frame – Methode

`AddControl`	Steuerelement einfügen
`RemoveControl`	Steuerelement entfernen

18.9 Multiseiten (MultiPage), Register (TabStrip)

Multipage – Eigenschaften

`Pages`	verweist auf das *Page*-Aufzählungsobjekt
`Pages(n)`	verweist auf ein einzelnes *Page*-Objekt
`MultiRow`	mehrere Zeilen mit Tabulatoren zur Blattauswahl
`TabOrientation`	Tabulatoren links / rechts / oben / unten

Page – Eigenschaften

`Caption`	Beschriftung der Seite (Tabulatortext)
`ScrollBars`	gibt an, ob Bildlaufleisten verwendet warden sollen
`KeepScrollBarsVisible`	Bildlaufleiste immer anzeigen
`ScrollWidth/-Height`	Grösse des verschiebbaren Bereichs
`ScrollLeft/-Top`	linke obere Ecke des sichtbaren Bereichs
`InsideWidth/-Height`	Grösse des nutzbaren Innenbereichs
`Zoom`	Vergrösserungsfaktor für Inhalt des Blatts
`TransitionEffect, -Period`	Effekt beim Wechsel zu einem anderen Blatt

18.10 Bildlaufleiste (ScrollBar) und Drehfeld (SpinButton)

ScrollBar, SpinButton - Eigenschaften

`Delay`	Verzögerung zwischen den Ereignissen in Millisekunden
`LargeChange`	seitenweise Änderung (nur für *ScrollBar*)
`Min/Max`	zulässiger Wertebereich
`Orientation`	Pfeile auf / ab oder links / rechts
`SmallChange`	Änderung beim Anklicken der Buttons
`Value`	aktueller Wert

ScrollBar, SpinButton – Ereignisse

`Change`	*Value* hat sich geändert
`Scroll`	Schiebefeld wird gerade bewegt (nur *ScrollBar*)
`SpinDown`	Pfeil nach unten (nach rechts) wurde gewählt (nur *SpinButton*)
`SpinUp`	Pfeil nach oben (nach links) wurde gewählt (nur *SpinButton*)

18.11 Anzeige (Image)

Image – Eigenschaften

`AutoSize`	Bildfeld passt sich an Bitmapgrösse an
`Border`	Umrandung
`Picture`	Bitmap
`PictureAlignment`	Ausrichtung der Bitmap (falls Bitmap kleiner als Bildfeld)
`PictureSizeMode`	Skalierung der Bitmap
`PictureTiling`	*True*, wenn Bitmap horizontal und vertikal wiederholt werden soll
`SpecialEffect`	3D-Effekt für Umrandung

Image – Ereignis

`Click`	das Steuerelement wurde angeklickt

18.12 Formelfeld (RefEdit)

RefEdit – Eigenschaften

`Value`	enthält eine Zeichenkette mit dem Zellbezug

RefEdit – Ereignisse

`Change`	*Value* hat sich geändert (das Ereignis tritt leider nicht immer auf)

18.13 Das UserForm – Objekt

`ActiveControl`	aktives Steuerelement innerhalb der Gruppe
`Controls`	Zugriff auf die enthaltenen Steuerelemente
`InsideWidth/-Height`	Grösse des nutzbaren Innenbereichs
`KeepScrollBarsVisible`	Bildlaufleiste immer anzeigen
`Picture`	Bitmap
`PictureAlignment`	Ausrichtung der Bitmap (falls Bitmap kleiner Bildfeld)
`PictureSizeMode`	Skalierung der Bitmap
`PictureTiling`	*True*, wenn Bitmap horizontal und vertical wiederholt warden soll
`SrollBars`	gibt an, ob Bildlaufleiste verwendet warden soll
`ScrollLeft/-Top`	linke obere Ecke des sichtbaren Bereichs
`ScrollWidth/-Height`	Grösse des verschiebbaren Bereichs
`Zoom`	Vergrösserungsfaktor für Inhalt des Rahmens

UserForm – Ereignisse

`Activate`	der Dialog wird angezeigt bzw. wieder aktiviert (nach *Deactivate*)
`Click`	der Dialog (nicht ein Steuerelement) wurde angeklickt
`Deactivate`	der Dialog verliert den Fokus, weil ein Subdialog angezeigt wird
`Initialize`	der Dialog wird in den Speicher geladen (Initialisierung)
`QueryClos`	der Dialog soll beendet werden (Schliessen-Button)
`Terminate`	der Dialog wird aus dem Speicher entfernt (Aufräumarbeiten)

19 Menüs und Symbolleisten

19.1 Diverses

Es gibt zwei Menüleisten

- Es gibt zwei voneinander unabhängige Menüleiste, die je nachdem, ob ein Tabellenblatt oder ein Diagramm aktiv ist, angezeigt werden. Wenn Sie ein neues Menükommando einfügen möchten, das in jedem Fall verfügbar ist, müssen Sie es in *beiden* Menüleisten einfügen.

Modus: ganzer Bildschirm

- Für den Modus "ganzer Bildschirm" werden eigene Einstellungen gespeichert.
- Name der Symbolleiste "Ganzer Bildschirm" = `Full Screen`
- Nachteil: Ist der Modus "Ganzer Bildschirm" aktiviert, sieht man erstens den unteren Pfeil der vertikale Bildlaufleiste nicht mehr, so dass man nicht mehr Zeilenweise sondern nur noch Seitenweise hinunterblättern kann, zweitens blendet es auch die Tabellenregister aus. Diese beiden Nachteile sind bei machen Anwendungen nicht akzeptabel, so dass auf eine andere Alternative augewichen werden muss.

Objekthierarchie

CommandBar[s]	Symbol- und Menüleisten, Kontextmenüs
CommandBarControls	Auflistung aller Einträge (Eigenschaft `Controls`)
CommandBarButton	Menükommando oder Symbol
CommandBarComboBox	Listenfeld
ComandBarPopup	Menü, Untermenü etc.
CommandbarControls	Auflistung aller Einträge (Eigenschaft `Controls`)
…	siehe oben

Wenn der Name für eine Symbolleiste schon vorhanden ist

- Wenn Sie versuchen, mit `Add` eine neue Symbolleiste mit einem bereits benutzten Namen zu erzeugen, kommt es zu einem Fehler. (Menü vermutlich schon einmal erstellt). Sie können den Fehler entweder mit `On Error Resume Next` abfangen (und in der nächstfolgenden Zeile `Err` auswerten), oder vor dem `Add`-Kommando in einer Schleife über alle Symbolleisten testen, ob schon eine mit dem gewünschten Name existiert.

Menü- und Symbolelemente (CommandBarControls)

- Der Zugriff auf diese Objekte erfolgt über die `CommandBarControls`-Aufzählung, die auf Objekte des Typs `CommandBarControl` zeigt. Der Zugriff auf diese Aufzählung erfolgt nicht wie sonst üblich durch eine gleichnamige Eigenschaft, sondern durch die Kurzform `Controls`. `CommandBarControl` ist ein Überobjekt, dessen Eigenschaften und Methoden davon abhängen, ob dadurch ein CommandBarButton-, ein CommandBarComboBox-, ein CommandBarPopup- oder ein anderes Objekt repräsentiert wird. Der Objekttyp kann durch die Eigenschaft `Type` ermittelt werden. (Obwohl im Office-Objektmodell nur die drei genannten Objekttypen vorgesehen sind, existieren in Wirklichkeit wesentlich mehr. Diese weiteren Objekttypen kommen in den eingebauten Menüs und Symbolleisten zum Einsatz, können aber nicht in eigenen Symbolleisten verwendet werden.)
- Mit `Controls.Add` können neue Symbole, Menüeinträge, Untermenüs oder Listenfelder in das Menü eingefügt werden. Dabei kann mit dem optionalen `Type`-Parameter der Typ

des Menüelements angegeben werden oder mit `Id` ein vordefiniertes Kommando verwendet werden.

```
Add Id:=123                          'vorgegebenes Kommando
Add Type:=msoControlButton           'Button bzw. Symbol (CommandBarButton)
Add Type:= msoControlEdit            'Testeingabefeld    (CommandBarComboBox)
Add Type:= msoControlDropdown        'Listenfeld         (CommandBarComboBox)
Add Type:= msoControlComboBox        'Kombinationslist.  (CommandBarComboBox)
Add Type:= msoControlPopup           'Menü/Untermenü     (CommandBarPopup)
```

- Wenn Sie mit `Add` den optionalen Parameter `Temporary:=True` verwenden, gilt der neue Menüeintrag als vorübergehend. Solche Einträge müssen beim Schliessen der Excel-Datei nicht explizit gelöscht werden, weil sie beim Verlassen von Excel automatisch wieder entfernt werden. (Einträge ohne `Temporary:=True` werden dagegen automatisch in der Datei `Excel.xlb` gespeichert

Symbole und Menüeinträge (CommandBarButton)

- Das sicherlich am häufigsten eingesetzte Menüelement ist `CommandBarButton`. Je nach Einstellung der *Style*-Eigenschaft (`msoButtonIcon`, `msoButtonCaption` oder `msoButtonIconAndCaption`) wird der Button als Symbol, als Text oder durch beides dargestellt.
- *Caption* bestimmt den angezeigten Text. Dieser Text wird bei Symbolen auch für den gelben Infotext (Quickinfo) verwendet, wenn nicht durch `TooltipText` ein anderer Text eingestellt ist. Für das Symbol gibt es keine Eigenschaft. Es kann nur durch die Methode `PasteFace` veränder werden. (Diese Methode setzt voraus, dass sich die Bitmap-Information eines Symbols in der Zwischenablage befindet. Diese Informationen können durch `CopyFace` von einem anderen Symbol in die Zwischenablage kopiert werden.)
- Mit der `OnAction`-Eigenschaft wird der Name der Ereignisprozedur angegeben, die beim Anklicken des Elements aufgerufen wird. (Das ist nur dann erforderlich, wenn die Ereignisse nicht über `OnClick`-Ereignisprozeduren verarbeitet werden sollen.)
- Wenn mehrere Menüeinträge oder Symbole zu einer Gruppe zusammengefasst werden sollen, kann durch `BeginGroup=True` ein Trennstrich oberhalb bzw. links vom Element angezeigt werden.

Text und Listenfelder (CommandBarComboBox)

- Text und Listenfelder können nur über den Programmcode erzeugt werden, nicht aber über den Anpassen-Dialog. (Nichtsdestotroz werden einmal per Code erzeugte Listenfelder aber mit allen Listeneinträgen in `Username8.xlb` gespeichert!)
- Es gibt drei verschiedene Typen dieses Felds, die alle durch das Objekt `CommanBarComboBox` repräsentiert werden (und sich durch unterschiedliche *Style*-Eigenschaften unterscheiden: `msoControlEdit`, `msoControlDropdown` oder `msoControlComboBox`). Eine kurze Beschreibung: *Edit* ist ein Texteingabefeld, `Dropdown` ist ein einfaches Listenfeld,. Die Kombination beider Felder ergibt `Combo`, also ein Listenfeld, in dem Sie zusätzlich zu den vorgegebenen Einträgen auch neue Text eingeben können (ganz wie beim Kombinationslistenfeld) Welchen Type Sie verwenden möchten, geben Sie im `Type`-Parameter von `Add` an.

```
Dim cmc As CommanBarComboBox
With CommandBars("…").Controls
  Set cbc = .Add(Type:=msoControlEdit)
```

```
  Set cbc = .Add(Type:=msoControlDropdown)
  Set cbc = .Add(Type:=msoControlComboBox)
End With
```

- Dem Listenfeld können mit `AddItem` neue Einträge hinzugefügt werden. `RemoveItem` entfernt einzelne Einträge, `Clear` alle Einträge. Die Eigenschaft `Text` gibt den aktuellen Inhalt des Textfelds bzw. die aktuelle Auswahl eines Listeneintrags an.
- Die Ereignisprozedur wird wie beim `CommandBarButton` durch die Eigenschaft `OnAction` eingestellt. An die Ereignisprozedur wird kein Parameter übergeben, d.h., es muss dort die Eigenschaft *Text* ausgewertet werden.
- Bei Symbolen zeitg Excel automatisch den `Caption`-Text als gelben Infotext an. Bei Listenfeldern wird aber aus Platzgründen oft gar kein `Caption`-Text eingestellt. In diesem Fall können Sie den Infotext auch mit `TooltipText` festlegen. (Diese Eigenschaft existiert für alle `CommandBar`-Elemente, wird aber selten verwendet. Sie hat Vorrang gegenüber enentuell ebenfalls verwendeten `Caption`-Texten.)

Menüs und Untermenüs (CommandBarPopup)

- Symbole und Menüeinträge können wahlweise direkt in eine Symbolleiste eingefügt werden oder in Menüs gruppiert werden. Innerhalb von Menüs sind wiederunm Untermenüs, darin Unteruntermenüs erlaubt. Das `CommanBarPopup`-Objekt dient dazu, Menüeinträge zu einer Gruppe zusammenzufassen. Der Zugriff auf die einzelnen Elemente erfolgt über die Eigenschaft `Controls`, die auf ein `CommanBarControls`-Objket verweist. Mit `Controls.Add` werden neue Menüelemente hinzugefügt. Das Menü wird mit `Caption` beschriftet.

Objektzugriff

Zugriff auf den Untermenüeintrag: Format / Blatt / Umenennen (CommandBarButton):

```
CommandBars("Workssheet Menu Bar"). _
Controls("Format").Controls("Blatt").Controls("Umbenennen")
```

- Statt des direkten Zugriffs auf ein Objekt können Sie sich in manchen Fällen auch der Methode `FindControl` des `CommanBar`-Objekts behelfen. Die Methode sucht nach dem ersten Objekt in einer Symbolleiste, das bestimmten Kriterien genügt. Die Kriterien sind für viele Anwendungsfälle allerdings unzureichend – so kann etwa nicht nach dem Namen eines Eintrags gesucht werden. Am ehesten kann `FindControl` für selbstdefinierte Elemente verwendet werden, wenn diesen Elementen eindeutige `Tag`-Eigenschaften zugewiesen wurden.

Die drei CommandBars-Typen

- Alle Leisten werden durch das Objekt CommandBars beschrieben. Die einzelnen Listen können Sie über die Eigenschaft *Type* unterscheiden.

Index	Konstante	Befehlsleiste
0	`msoBarTypeNormal`	Symbolleiste
1	`msoBarTypeMenuBar`	Menüleiste
2	`msoBarTypePopup`	Kontextmenü

Liste mit Übersetztung aller Leisten

- Als Index vom `CommandBars` wird der englische Name der Menü- oder Symbolleiste verwendet. `CommandBars("Worksheet Menu Bar")`

Index	Typenummer	Typ	Englischer Name	Deutscher Name
1	1	Menüleiste	Worksheet Menu Bar	Arbeitsblatt-Menüleiste
2	1	Menüleiste	Chart Menu Bar	Diagrammmenüleiste
3	0	Symbolleiste	Standard	Standard
4	0	Symbolleiste	Formatting	Format
5	0	Symbolleiste	PivotTable	PivotTable
6	0	Symbolleiste	Chart	Diagramm
7	0	Symbolleiste	Reviewing	Überarbeiten
8	0	Symbolleiste	Forms	Formular
9	0	Symbolleiste	Stop Recording	Aufzeichnung beenden
10	0	Symbolleiste	External Data	Externe Daten
11	0	Symbolleiste	Formula Auditing	Formelüberwachung
12	0	Symbolleiste	Full Screen	Ganzer Bildschirm
13	0	Symbolleiste	Circular Reference	Zirkelverweis
14	0	Symbolleiste	Visual Basic	Visual Basic
15	0	Symbolleiste	Web	Web
16	0	Symbolleiste	Control Toolbox	Steuerelement-Toolbox
17	0	Symbolleiste	Exit Design Mode	Entwurfsmodus beenden
18	0	Symbolleiste	Refresh	Aktualisieren
19	0	Symbolleiste	Watch Window	Überwachungsfenster
20	0	Symbolleiste	PivotTable Field List	PivotTable-Feldliste
21	0	Symbolleiste	Borders	Rahmenlinien
22	0	Symbolleiste	Protection	Schutz
23	0	Symbolleiste	Text To Speech	Text zu Sprachein- und -ausgabe
24	0	Symbolleiste	Drawing	Zeichnen
25	0	Symbolleiste	EuroPlaceholder_2000	EuroPlaceholder_2000
26	0	Symbolleiste	EuroValue	EuroValue
27	2	Kontextmenü	Query and Pivot	Abfragen und pivotieren
28	2	Kontextmenü	PivotChart Menu	PivotChart-Menü
29	2	Kontextmenü	Workbook tabs	Arbeitsmappen-Registerkarte
30	2	Kontextmenü	Cell	Zelle
31	2	Kontextmenü	Column	Spalte
32	2	Kontextmenü	Row	Zeile
33	2	Kontextmenü	Cell	Zelle
34	2	Kontextmenü	Column	Spalte
35	2	Kontextmenü	Row	Zeile
36	2	Kontextmenü	Ply	Blatt
37	2	Kontextmenü	XLM Cell	XLM-Zelle
38	2	Kontextmenü	Document	Dokument
39	2	Kontextmenü	Desktop	Desktop
40	2	Kontextmenü	Nondefault Drag and Drop	Nicht standardmäßiges Drag & Drop
41	2	Kontextmenü	AutoFill	AutoAusfüllen
42	2	Kontextmenü	Button	Schaltfläche
43	2	Kontextmenü	Dialog	Dialogfeld
44	2	Kontextmenü	Series	Reihen
45	2	Kontextmenü	Plot Area	Bereich zeichnen
46	2	Kontextmenü	Floor and Walls	Boden und Wände
47	2	Kontextmenü	Trendline	Trendlinie
48	2	Kontextmenü	Chart	Diagramm
49	2	Kontextmenü	Format Data Series	Datenreihen formatieren
50	2	Kontextmenü	Format Axis	Achse formatieren
51	2	Kontextmenü	Format Legend Entry	Legendeneintrag formatieren
52	2	Kontextmenü	Formula Bar	Bearbeitungsleiste

Index	Typenummer	Typ	Englischer Name	Deutscher Name
53	2	Kontextmenü	PivotTable Context Menu	PivotTable-Kontextmenü
54	2	Kontextmenü	Query	Abfrage
55	2	Kontextmenü	Query Layout	Abfrageentwurf
56	2	Kontextmenü	AutoCalculate	AutoBerechnen
57	2	Kontextmenü	Object/Plot	Objekt/Zeichnen
58	2	Kontextmenü	Title Bar (Charting)	Titelleiste (Diagramm)
59	2	Kontextmenü	Layout	Layout
60	2	Kontextmenü	Pivot Chart Popup	PivotChart-Popupmenü
61	2	Kontextmenü	Phonetic Information	Phonetische Information
62	2	Kontextmenü	Auto Sum	Automatische Summenfunktion
63	2	Kontextmenü	Paste Special Dropdown	Dropdown: Einfügen Extras
64	2	Kontextmenü	Find Format	Format suchen
65	2	Kontextmenü	Replace Format	Format ersetzen
66	0	Symbolleiste	WordArt	WordArt
67	0	Symbolleiste	Picture	Grafik
68	0	Symbolleiste	Shadow Settings	Schatteneinstellungen
69	0	Symbolleiste	3-D Settings	3D-Einstellungen
70	0	Symbolleiste	Drawing Canvas	Zeichnungsbereich
71	0	Symbolleiste	Organization Chart	Organigramm
72	0	Symbolleiste	Diagram	Diagramm
73	0	Symbolleiste	Borders	Rahmen
74	0	Symbolleiste	Borders	Rahmenlinien
75	0	Symbolleiste	Draw Border	Rahmenlinie zeichnen
76	0	Symbolleiste	Chart Type	Diagrammtyp
77	0	Symbolleiste	Pattern	Muster
78	0	Symbolleiste	Font Color	Schriftfarbe
79	0	Symbolleiste	Fill Color	Füllfarbe
80	0	Symbolleiste	Line Color	Linienfarbe
81	0	Symbolleiste	Order	Reihenfolge
82	0	Symbolleiste	Nudge	Präzisionsausrichtung
83	0	Symbolleiste	Align or Distribute	Ausrichten oder verteilen
84	0	Symbolleiste	Rotate or Flip	Drehen oder kippen
85	0	Symbolleiste	Lines	Linien
86	0	Symbolleiste	Connectors	Verbindungen
87	0	Symbolleiste	AutoShapes	AutoFormen
88	0	Symbolleiste	Callouts	Legenden
89	0	Symbolleiste	Flowchart	Flussdiagramm
90	0	Symbolleiste	Block Arrows	Blockpfeile
91	0	Symbolleiste	Stars & Banners	Sterne & Banner
92	0	Symbolleiste	Basic Shapes	Standardformen
93	0	Symbolleiste	Insert Shape	Form einfügen
94	2	Kontextmenü	Shapes	Formen
95	2	Kontextmenü	Inactive Chart	Inaktives Diagramm
96	2	Kontextmenü	Excel Control	Excel-Steuerelement
97	2	Kontextmenü	Curve	Krümmen
98	2	Kontextmenü	Curve Node	Knoten krümmen
99	2	Kontextmenü	Curve Segment	Kurvenabschnitt
100	2	Kontextmenü	Pictures Context Menu	Grafikkontextmenü
101	2	Kontextmenü	OLE Object	OLE-Objekt
102	2	Kontextmenü	ActiveX Control	ActiveX-Steuerelement
103	2	Kontextmenü	WordArt Context Menu	WordArt-Kontextmenü
104	2	Kontextmenü	Rotate Mode	Drehungsmodus
105	2	Kontextmenü	Connector	Verbindung
106	2	Kontextmenü	Script Anchor Popup	Skriptanchor-Popupmenü

Index	Typenummer	Typ	Englischer Name	Deutscher Name
107	2	Kontextmenü	Canvas Popup	Canvas Popup
			Organization Chart	
108	2	Kontextmenü	Popup	Organization Chart Popup
109	2	Kontextmenü	Diagram	Diagramm
110	2	Kontextmenü	Select	Markieren
111	2	Kontextmenü	Layout	Layout
112	0	Symbolleiste	Task Pane	Aufgabenbereich
113	2	Kontextmenü	Built-in Menus	Integrierte Menüs
114	2	Kontextmenü	System	System
115	0	Symbolleiste	Clipboard	Zwischenablage
116	0	Symbolleiste	Envelope	Umschlag
117	0	Symbolleiste	Online Meeting	Onlinebesprechung

Makro, das die obige Liste mit allen Arten von Leisten erstellen

```
Dim i As Integer
Dim sIndex As Integer
Dim sType As String
Cells.Clear
Range(”A1:E1”).Font.Bold = True
Range(”A1”).Value = ”Index”
Range(”B1”).Value = ”Typenummer”
Range(”C1”).Value = ”Typ”
Range(”D1”).Value = ”Englischer Name”
Range(”E1”).Value = ”Deutscher Name”
 For i = 1 To CommandBars.Count
  Select Case CommandBars(i).Type
   Case 0
      sIndex = 0
      sType = ”Symbolleiste”
   Case 1
      sIndex = 1
      sType = ”Menüleiste”
   Case 2
     sIndex = 2
     sType = ”Kontextmenü”
   Case Else
      s = ”nicht ermittelbar”
   End Select
   Range(”A2”).Select
   If ActiveCell.Value = ”” Then
     ActiveCell.Value = i
     ActiveCell.Offset(0, 1).Value = sIndex
     ActiveCell.Offset(0, 2).Value = sType
     ActiveCell.Offset(0, 3).Value = CommandBars(i).Name
     ActiveCell.Offset(0, 4).Value = CommandBars(i).NameLocal
   ElseIf ActiveCell.Offset(1, 0).Value = ”” Then
     ActiveCell.Offset(1, 0).Select
     ActiveCell.Value = i
     ActiveCell.Offset(0, 1).Value = sIndex
     ActiveCell.Offset(0, 2).Value = sType
     ActiveCell.Offset(0, 3).Value = CommandBars(i).Name
     ActiveCell.Offset(0, 4).Value = CommandBars(i).NameLocal
   Else: Selection.End(xlDown).Select
      ActiveCell.Offset(1, 0).Select
      ActiveCell.Value = i
```

```
    ActiveCell.Offset(0, 1).Value = sIndex
    ActiveCell.Offset(0, 2).Value = sType
    ActiveCell.Offset(0, 3).Value = CommandBars(i).Name
    ActiveCell.Offset(0, 4).Value = CommandBars(i).NameLocal
  End If
Next i
Columns(”A:E”).EntireColumn.AutoFit
Range(”A1”).Select
```

19.2 Oberflächengestaltung für eigenständige Excel-Anwendungen

Erweiterung des Standardmenüs

- Vorteil: Nach dem Laden der Datei können auch andere Excel-Dateien problemlos weiterbearbeitet werden. Nachteil: die neuen Bedienungselemente gehen zwischen den normalen Menüs, Symbolleisten etc. unter.

Eigene Symbolleiste ein- und ausblenden

- Vorteil: Es gibt eine klare Trennung zwischen "normalen" Excel-Elementen und zusätzlichen Kommandos. Nachteil: Die Symbolleiste beansprucht zusätzlich Platz am Bildschirm.

Eigenes Standardmenü verwenden

- Vorteil: Es stehen wirklich nur die Kommandos zur Verfügung, die der Anwender zur Bedienung momantan benötigt. Das ist insbesondere für Anwender praktisch, die keine Excel-Profis sind und daher durch die Fülle der Excel-Menüeinträge überfordert sind. (Ausserdem mindert das die Gefahr, unbeabsichtigt Kommandos auszuführen, die Schaden anrichten können.) Der Nachteil: Excel ist kaum wiederzuerkennen; plötzlich fehlen normale Menüs uns Symbolleisten. Ein "normales" Weiterarbeiten mit anderen Dateien ist nur möglich, wenn bei einem Fensterwechsel automatisch die Standardkonfiguration wiederhergestellt wird.

Es folgen die drei obigen Varianten ausführlich beschrieben:

Erweiterung des Standardmenüs

Menüs kopieren

- Wenn Sie ein zusätzliches Menü in der Standardmenüleiste verwenden möchten, bestehen dazu zwei Möglichkeiten:
- Entweder Sie führen alle Erweiterungen am Standardmenü durch zahllose VBA-Anweisungen durch, die neue Einträge erstellen, beschriften und Ereignisprozeduren zuordnen.
- Oder Sie speichern Ihr neues Menü in einer angebundenen Symbolleiste. Beim Start Ihres Programms lassen Sie diese Symbolleiste unsichtbar, kopieren das Menü aber in die Standardmenüleiste. Das `CommandBarControl`-Objekt sieht dazu die `Copy`-Methode vor. Durch das folgende Kommando wird das angegebene Objekt – ein einzelner Menüeintrag aber auch ein ganzes Menü (`CommandBarPopup`) – vor der durch `position` angegebenen Stelle in die Zielsymbolleiste kopiert.

```
quellobjekt.Copy Zielsymbolleiste, positon
```

- Die zweite Variante hat den Vorteil, dass der Menüentwurf interaktiv erfolgen kann und viel weniger Code erforderlich ist. Dieser Abschnitt beschränkt sich daher auf diese Variante.
- In beiden Fällen müssen Sie vorher testen, ob sich das neue Menü nicht bereits in der Standardmenüleisete befindet! Andernfalls kann es passieren, dass Ihr Menü doppelt erscheint. Ausserdem sollten Sie sich darum kümmern, dass das Menü beim Schliessen der Datei wieder entfernt wird.

Beispielprogramm

- An die Beispieldatei ist die Symbolleiste "CommandBar-Copy" angebunden. In `Workbook_Open` wird das erste Menü dieser Symbolleiste mit `Copy` an die vorletzte

Stelle der Standardmenüs kopiert. Das Menü wird mit `Visible=True` sichtbar, die zugrundeliegende Symbolleiste dagegen mit `Visible=False` unsichtbar gemacht.

```
Private Sub Workbook_Open()
  Dim standardmenubar As CommandBar
  Dim mycommandbar As CommandBar
  Dim c As CommandBarControl
  Set standardmenubar = _
    Application.CommandBars(”worksheet menu bar”)
  Set mycommandbar = Application.CommandBars(”CommandBar-Copy”)
  mycommandbar.Visible = False
  ' Test, ob Menü schon existiert
  For Each c In standardmenubar.Controls
    If c.Caption = mycommandbar.Controls(1).Caption Then
      c.Visible = True
      Exit Sub
    End If
  Next
  ' Menü existiert noch nicht: daher kopieren
  Set c = mycommandbar.Controls(1).Copy _
    (standardmenubar, standardmenubar.Controls.Count)
  c.Visible = True
End Sub

Private Sub Workbook_Activate()
  Application.CommandBars(”worksheet menu bar”).Controls(”Neues
Menü”).Visible = True
End Sub

Private Sub Workbook_Deactivate()
  On Error Resume Next
  Application.CommandBars(”worksheet    menu    bar”).Controls(”Neues
Menü”).Visible = False
End Sub
```

- Beim Schliessen der Datei wird das Menü aus der Standardmenüleiste entfernt. Ausserdem wird die Symbolleiste gelöscht, so dass sie nicht in Excel.xlb gespeichert wird.

```
' Aufräumarbeiten
Private Sub Workbook_BeforeClose(Cancel As Boolean)
  Dim standardmenubar As CommandBar
  Dim mycommandbar As CommandBar
  Dim c As CommandBarControl
  Set standardmenubar = _
    Application.CommandBars(”worksheet menu bar”)
  Set mycommandbar = Application.CommandBars(”CommandBar-Copy”)
  ' Test, ob Menü schon existiert
  For Each c In standardmenubar.Controls
    If c.Caption = mycommandbar.Controls(1).Caption Then
      c.Delete
    End If
  Next
  mycommandbar.Delete
End Sub
```

Eigene Symbolleiste ein- und ausblenden

Symbolleiste anbinden

- Symbolleistenbereich / Kontextmenü: Anpassen / Register: Symboleiste / Kopf: Anfügen. Die Symbolleiste gehört danach zur aktuellen Arbeitsmappe. Sie können aber keine Änderungen an den vordefinierten Menüs und Symbolleisten in einer Excel-Datei speichern.

Eigene Symbolleiste ein- und ausblenden

- Diese Variante ist mit dem geringten Aufwand verbunden. In der Beispieldatei wird die Symbolleiste beim Laden sichtbar gemacht, beim Schliessen wieder gelöscht. Ausserdem wird die Symbolleiste automatisch ein- und ausgebledet, je nach dem, ob gerade ein Fenster der Arbeitsmappe oder ein Fenster einer anderen Arbeitsmappe sichtbar ist.

```
Private Sub Workbook_Open()
  Application.CommandBars(”Commandbar-Auto”).Visible = True
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
  Application.CommandBars(”Commandbar-Auto”).Delete
End Sub

Private Sub Workbook_Activate()
  Application.CommandBars(”Commandbar-Auto”).Visible = True
End Sub

Private Sub Workbook_Deactivate()
  On Error Resume Next
  Application.CommandBars(”Commandbar-Auto”).Visible = False
End Sub
```

Eigenes Standardmenü verwenden

- Die Beispieldatei verfolgt eine ähnliche Strategie wie *CommandBar-Copy.xls*: In der Symbolleiste ”CommandBar-New”, die nie angezeigt wird, werden die Menüeinträge für das Hauptmenü gespeichert. Beim Laden der Datei wird eine neue Menüleiste ”NewMenü” erzeugt, in die der Inhalt von ”CommandBar-New” kopiert wird. Sobald diese neue Menüleiste sichtbar gemacht wird, verschwindet die Standardmenüleiste. Beim Programmende wird sowohl die Symbolleiste als auch die Menüleiste wieder gelöscht, das Standardmenü erscheint automatisch wieder.

```
Private Sub Workbook_Open()
Dim cb As CommandBar, c As CommandBarControl
  On Error Resume Next
  ' Symbolleiste mit Basisdaten unsichtbar
  Application.CommandBars(”Commandbar-New”).Visible = False
  Set cb = Application.CommandBars.Add(Name:=”NewMenu”, _
    MenuBar:=True, Position:=msoBarTop)
  For Each c In Application.CommandBars(”Commandbar-New”).Controls
    c.Copy cb
  Next
End Sub

' Aufräumarbeiten
Private Sub Workbook_BeforeClose(Cancel As Boolean)
  On Error Resume Next
  ' Symbolleiste löschen
```

```
  Application.CommandBars("Commandbar-New").Delete
  ' neue Menüleiste löschen
   (dabei wird automatisch die Standardmenü aktiv)
  Application.CommandBars("NewMenu").Delete
End Sub
```

- Die beiden Prozeduren `Workbook_Activate` und `Workbook_Deactivate` sorgen auch in diesem Programm dafür, dass das Menü verschwindet, wenn in eine andere Date gewechselt wird. Sobald die Arbeitsmappe wieder aktiviert wird, erscheint auch das Menü wieder.
- Neu im Vergleich zu den beiden vorangegangenen Programmen sit der Umgang mit den Symbolleisten: Solange die Beispieldatei aktiv ist, werden alle Symbolleisten ausgeblendet (es ist also wirklich nur das neue Menü sichtbar). Das Programm speichert beim Ausblenden die Liste aller sichtbaren Symbolleisten in der `Collection`-Variable `visibleCommandBars`. Alle Symbolleisten dieser Liste werden automatisch wieder sichtbar gemacht, sobald ein anderes Tabellenblatt aktiviert wird.

```
Private Sub Workbook_Activate()
  Dim cb As CommandBar
  ' neue Menüleiste sichtbar
  Application.CommandBars("NewMenu").Visible = True
  ' alle Symbolleisten abschalten
  For Each cb In Application.CommandBars
    If cb.Type = msoBarTypeNormal And cb.Visible = True Then
      visibleCommandBars.Add cb, cb.Name
      cb.Visible = False
    End If
  Next
End Sub

Private Sub Workbook_Deactivate()
  Dim cb As Object
  On Error Resume Next
  Application.CommandBars("NewMenu").Visible = False
  ' alle Symbolleisten abschalten
  For Each cb In visibleCommandBars
    cb.Visible = True
  Next
  Set visibleCommandBars = Nothing
End Sub
```

19.3 Die Programmierung von Menüs

Adaptives Menü ausschalten

```
If Application.CommandBars.AdaptiveMenus = True _
Then Application.CommandBars.AdaptiveMenus = False
```

Alle Leisten und Kontextmenüs ausblenden

```
Dim cb As CommandBar
  For Each cb In Application.CommandBars
  cb.Enabled = False
Next cb
```

Alle Leisten und Kontextmenüs einblenden

```
Dim cb As CommandBar
  For Each cb In Application.CommandBars
  cb.Enabled = True
Next cb
```

Menüleiste ausblenden

```
Application.CommandBars(1).Enabled = False
```

Menüleiste einblenden

```
Application.CommandBars(1).Enabled = True
```

Ein eigenes Menü erzeugen

```
Dim i As Integer
Dim i_Hilfe As Integer
Dim MenüNeu As CommandBarControl
  i = Application.CommandBars(1).Controls.Controls
  i_Hilfe = Application.CommandBars(1).Controls(i).Index
  Set MenüNeu = Application.CommandBars(1).Controls.Add _
    (Type:=msoControlPopup, _
    Before:=i_Hilfe,
    Temporary:=True)
  Menü.Caption = ”&Eigene Funktionen”
```

- Definieren Sie im erste Schritt zwei Integer-Variablen, die zum einen die Anzahl der Menüs ermittelt, die momentan in der Arbeitsplatz-Menüleiste eingebunden sind, und zum anderen die Position des Hilfemenüs ermitteln. Eine weitere Objektvariabel vom Typ CommandBarControl wird gebraucht, um den neuen Menüpunkt einzufügen. Über die Methode Count zählen Sie die Anzahl der Menüs in der Arbeitsblatt-Menüleiste und speichern Sie in der Variablen i. Im nächsten Schritt ermitteln Sie die Position des Hilfe-Menüs, welches standardmässig ganz rechts in der Arbitsblatt-Menüleiste steht. Die Arbeitsblatt-Menüleiste können Sie direkt über das Objekt CommandBars(1) ansprechen. Über die Eigenschaft Controls bekommen Sie alle Steuerelemente der angegebenen Menüleiste.

Syntax der Methode Add

- Mit Hilfe der Methode Add fügen Sie ein neues Menü ein. Die Methode Add hat die Syntax:

```
Add(Type, Id, Before, Temporary)
```

- Beim Argument `Type` geben Sie an, um welche Art Steuerelement es sich dabei handeln soll. Zur Auswahl stehen die Konstanten aus der folgenden Tabelle

Konstante	Beschreibung
`msoControlButton`	Fügt Schaltflächenelement ein
`msoControlEdit`	Fügt ein Eingabefeld ein
`msoControlDropdown`	Fügt ein Dropdown-Feld ein
`msoControlComboBox`	Fügt ebenso ein Dropdown-Feld ein
`msoControlPopup`	Fügt ein Dropdown-Menü ein

- Beim Argument `ID` können Sie sich entscheiden, ob Sie zusätzlich zum Menütext auch noch ein Symbol anzeigen möchten. Dieses Argument funktioniert jedoch nur innerhalb eines Menüs, also für einen Menübefehl.
- Mit dem Argument `Before` legen Sie die genaue Position des Menüs fest.
- Setzen Sie das Argument `Temporary` auf den Wert `True`, wenn das neue Steuerelement temporär sein soll.

Arbeitsblatt Menüleiste zurücksetzen

```
Application.CommandBars(1).Reset
```

Gezielt das neue Menü löschen

```
On Error Resume Next
  With Application.CommandBars(1)
    .Controls(”&Eigene Funktionen”).Delete
  End with
End Sub
```

Menübefehle im neuen Menü einfügen

```
Dim i As Integer
Dim i_Hilfe As Integer
Dim MenüNeu As CommandBarControl
Dim MB As CommandBarControl
  i = Application.CommandBars(1).Controls.Controls
  i_Hilfe = Application.CommandBars(1).Controls(i).Index
  Set MenüNeu = Application.CommandBars(1).Controls.Add _
    (Type:=msoControlPopup, _
    Before:=i_Hilfe,
    Temporary:=True)
  Menü.Caption = ”&Eigene Funktionen”

  Set MB = MenüNeu.Controls.Add(Type:=msoControlButton)
    With MB
      .Caption = ”&Formelzellen markieren”
      .Style = msoButtonCaption
      .OnAction = ”FormelMarkieren”
      .FaceID = 3
      .BeginGroup = True
    End With
```

- Über die Eigenschaft `FaceId` können Sie dem Menübefehl auch noch ein Symbol hinzufügen. Allerdings muss dabei dann die Eigenschaft `Styles` mit der Konstante `msoButtonAndCaption` angegeben werden.
- `BeginGroup` für Trennlinie
- Mit `OnAction` Makro angeben

Einen neuen Menüeintrag ins Menü Extras einfügen

```
Dim cbb As CommandBarButton
Set cbb = _
  Application.CommandBars(”Worksheet Menu Bar”). _
    Controls(”Extras”).Controls.Add()
cbb.Caption  ”Ein neues Kommando”
cbb.BeginGroup = True
cbb.OnAction = ”Makro1”
```

Den neuen Menüeintrag im Menü Extras wieder löschen

```
Application.CommandBars(”Worksheet Menu Bar”). _
  Controls(”Extras”). Controls(”Ein neues Kommando”).Delete
```

Alle Namen der Menüleistenmenüs ausgeben

```
Dim ctrl As CommandBarControl
Dim i As Integer
  For Each ctrl In Application.CommandBars(1).Controls
    i = i + 1
    Debug.Print ”Ctrlpunkt ” & i & ” ---> ” & ctrl.Caption
  Next ctrl
```

Alle Menübefehle im Menü Format deaktivieren

```
Dim ctrl As CommandBarControl
For Each ctrl In Application.CommandBars(”Format”).Controls
    ctrl.Enabled = False
Next ctrl
```

Alle Menübefehle im Menü Format aktivieren

```
Dim ctrl As CommandBarControl
For Each ctrl In Application.CommandBars(”Format”).Controls
    ctrl.Enabled = True
Next ctrl
```

Bestimmte Menübefehle suchen mit der Methode FindControl

Syntax der Methode FindControl

```
FindControl(Type, Id, Tag, Visible, Recursive)
```

- Mit dem Argument `Type` legen Sie den Type des Steuerelements fest, nach dem gesucht werden soll. Dabei können Sie u.a. aus folgenden Typkonstanten auswählen:

Konstante	Beschreibung bzw. Beispiel
`msoControlButton`	normale Symbolschaltfläche
`msoControlButtonPopup`	mehrstufiger Menübefehl
`msoControlPopup`	ein Menü (Date, Format, Ansicht)
`msoControlDropdown`	das Dropdown, um die Ansicht zu vergrössern
`msoControlExpandingGrid`	das Dropdown aus der Symbolleiste ”Format”, um einen

	Rahmen um einen Zellbereich zu legen.
msoControlGraphicCombo	das Dropdown aus der Symbolleiste Format, um die Schaltfläche einzustellen
msoControlGrid	die Farbpalette aus der Symbolleiste ”Format”, um die Schriftfarbe bzw. die Füllfarbe festzulegen

Makro, das die Ids der Arbeitsblattmenüleiste ermittelt (Datei, Format, …)

- Das Argument `Id` stellt einen Index dar, welcher das Menü bzw. den Menübefehl eindeutig identifiziert. Das folgende Makro schriebt die `Id` für alle Menüs der Arbeitsblatt-Menüleiste in der Direktbereich:

```
Dim Menüleiste As CommandBar
Dim i As Integer
Dim n As Integer
  Set Menüleiste = CommandBars(1)
  n = Menüleiste.Controls.Count
  For i = 1 To n
    Debug.Print Menüleiste.Controls(i).ID & _
      ” ---> ” & Menüleiste.Controls(i).Caption
  Next
```

Makros, das die Ids aus dem Menü Extra ermittelt

```
Dim Menüleiste As CommandBar
Dim ctrl As CommandBarControl
Dim i  As Integer
  Set Menüleiste = CommandBars(1)
  i = 1
  For Each ctrl In Menüleiste.Controls(”Extras”).Controls
    Debug.Print ctrl.ID & ” ---> ” & ctrl.Caption
      i = i + 1
  Next ctrl
```

- Mit Hilfe des Arguments `Tag` können Sie nach der Beschriftung des Menüs bzw. des Menübefehls suchen.
- Das Argument `Visible` ist optional und liefert den Wert `True`, wenn in die Suche nur sichtbare Befehlsleisten-Steuerelemente einbezogen werden sollen.
- Das Argument `Recursive` setzen Sie auf den Wert `True`, wenn die Befehlsleiste und alle zugehörigen Untersymbolleisten in die Suche eingeschlossen werden sollen

- Kann die Methode `FindControl` kein Steuerelement finden, welches Sie in den Kriterien angegeben haben, dann meldet die Mehtode den Wert `Noting` zurück.

Makro, das mit der Methode FindControl das Menü Extras findet und den Befehl ”Schützen” deaktiviert

- Variante 1

```
Dim Menü As CommandBarPopup
Dim ctrl As CommandBarControl
  Set Menü = Application.CommandBars.FindControl _
   (Type:=msoControlPopup, ID:=30007)
  For Each ctrl In Menü.Controls
    If ctrl.Caption = ”S&chutz” Then ctrl.Enabled = False: Exit Sub
 Next ctrl
 Set Menü = Nothing
```

- Variante 2 (Direkt über die ID des Untermenübefehls zugreifen)

```
Dim ctrl As CommandBarPopup
 Set ctrl = Application.CommandBars.FindControl(ID:=30029)
 If ctrl Is Nothing Then Else ctrl.Enabled = False
```

- Variante 3 (Ohne Methode FindControl)

```
Application.CommandBars(1).Controls(”Extras”).Controls(”Schutz”). _
Enabled = False
```

- Diese Lösung hat den Nachteil, dass Sie darauf achten müssen, ob Menübefehle geändert werden bzw. wenn Sie mehrsprachige Anwender haben. Das Argument `ID` bei der Methoe `FindControl` bleibt beispielsweise für das Menü Extras” immer konstant, während sich die Beschriftung des Menüs änder kann.

Kaskadenmenüs erstellen

- Möchten Sie mehrere Menübefehle miteinander verschachteln, um einzelne Befehle damit zu gruppieren, erstellen Sie ein Kaskaden-Menü. So können Sie z.B. im Menü ”Bearbeiten” ganz unten ein neues Kaskaden-Menü einfügen.

```
Dim Kaskade As CommandBarPopup
Dim UnterMenü As CommandBarPopup
Dim UKaskade As CommandBarControl
Dim i As Integer

‘Gruppierungsbefehl
    i = CommandBars(1).Controls(”Bearbeiten”).Controls.Count
    Set Kaskade = CommandBars(1).Controls(”Bearbeiten”). _
       Controls.Add(Type:=msoControlPopup, before:=i, _
         Temporary:=True)
    With Kaskade
        .Caption = ”&Markieren Zellen mit ...”
        .BeginGroup = True
    End With

’Unterbefehl
    With Kaskade.Controls.Add(Type:=msoControlButton, _
      Temporary:=True)
        .Caption = ”&Formeln”
        .OnAction = ”FormelZellenMarkieren”
        .Style = msoButtonIconAndCaption
        .FaceId = 373
        .Enabled = True
    End With

’Unterbefehl
    With Kaskade.Controls.Add(Type:=msoControlButton, _
      Temporary:=True)
        .Caption = ”&Kommentaren”
        .OnAction = ”KommentarZellenMarkieren”
        .FaceId = 1589
        .Style = msoButtonIconAndCaption
        .Enabled = True
```

```
    End With

'Wieder Guppierungsbefehl
    Set UnterMenü = Kaskade.Controls.Add(Type:=msoControlPopup, _
      Temporary:=True)
    With UnterMenü
        .Caption = "&Gütigkeitsregeln"
        .BeginGroup = True
    End With

'Unter Unterbefehl
    Set UKaskade = UnterMenü.Controls.Add(Type:=msoControlButton)
    With UKaskade
        .Caption = "&Alle Gültigkeitsregeln"
        .OnAction = "GültigkeitsZellenMarkieren"
    End With

'Unter Unterbefehl
    Set UKaskade = UnterMenü.Controls.Add _
      (Type:=msoControlButton)
    With UKaskade
        .Caption = "gleich&e Gültigkeitsregeln"
        .OnAction = "GleicheGültigkeitsZellenMarkieren"
    End With

'Unterbefehl
 With Kaskade.Controls.Add(Type:=msoControlButton, Temporary:=True)
    .Caption = "&bedingten Formaten"
    .OnAction = "BedingtFormatierteZellenMarkieren"
    .Style = msoButtonIconAndCaption
    .Enabled = True
    .BeginGroup = True
    .FaceId = 962
  End With
  Set Kaskade = Nothing
  Set UKascade = Nothing
  Set UnterMenü = Nothing
  Application.CommandBars(1).Controls("Bearbeiten").Execute
End Sub
```

- Mit `msoControlButton` als Argument von `Type`, legen Sie fest, dass keine weiteren Untermenüs für diesen Menübefehl mehr folgen.
- Mit `msoControlPopup` als Argument von `Type` legen sie fest, dass weitere Untermenüs für diesen Menübefehl folgen.
- Mit der Eigenschaft `Style` sagen Sie aus, wie der Menübefehl angezeigt werden soll. Entscheiden Sie sich für die Konstante `msoButtonIconAndCaption`**,** um sowohl ein Symbol als auch eine Beschriftung anzuzeigen.
- Geben Sie zum Abschluss die Objektvariable, für die im Arbeitsspeicher Platz reserviert wurde, wieder frei, indem Sie die einzelnen Variablen auf den Wert `Noting` setzen.
- Wenn Sie ein Menü automatisch aufklappen möchten, setzen Sie die Methode `Execute` ein.

Die passenden Makros dazu

```
Sub FormelZellenMarkieren()
```

```
On Error GoTo fehler
  Selection.SpecialCells(xlCellTypeFormulas).Select
  Exit Sub
fehler:
  MsgBox ”Es gibt auf diesem Blatt keine Zellen mit Formeln!”
End Sub

Sub KommentarZellenMarkieren()
On Error GoTo fehler
  Selection.SpecialCells(xlCellTypeComments).Select
  Exit Sub
fehler:
  MsgBox ”Es gibt auf diesem Blatt keine Zellen mit Kommentaren!”
End Sub

Sub GültigkeitsZellenMarkieren()
On Error GoTo fehler
  Selection.SpecialCells(xlCellTypeAllValidation).Select
  Exit Sub
fehler:
  MsgBox ”Es gibt auf diesem Blatt keine Zellen mit
Gültigkeitsregeln!”
End Sub

Sub GleicheGültigkeitsZellenMarkieren()
On Error GoTo fehler
  Selection.SpecialCells(xlCellTypeSameValidation).Select
  Exit Sub
fehler:
  MsgBox ”Es gibt auf diesem Blatt keine weiteren Zellen mit
denselben Gültigkeitsregeln!”
End Sub

Sub BedingtFormatierteZellenMarkieren()
On Error GoTo fehler
  Selection.SpecialCells(xlCellTypeAllFormatConditions).Select
  Exit Sub
fehler:
  MsgBox ”Es gibt auf diesem Blatt keine Zellen mit bedingten
Formaten!”
End Sub
```

ID oder FaceID?

- Die Eigenschaft `FaceId` bestimmt das Aussehen, jecoh nicht die Funktion einer Befehlsleisten-Schaltfläche. Die Eigenschaft `ID` des `CommandBarControl`-Objekt bestimmt die Funktion der Schaltfläche

Menüeinträge mit Auswahlhäcken

- Unter den vielen vordefinierten Symbolen von Excel gibt es auch eines, das ein Auswahlhäkchen darstellt `(Id=849)`. Allerdings ist es nicht möglich, via Anpassen-Dialog interaktiv einen neuen Menüeintrag mit diesem Symbol zu erzeugen. Auch eine Veränderung der `Id`-Eigenschaft eines vorhandenen Menüeintrags ist nicht zulässig (die Eingenschaft darf nur gelesen werden). Daher muss ein Menüeintrag mit Häckhen beeim Programmstart per VBA-Code erzeugt werden

Zugriff auf Menü innerhalb der Symbolleiste CommandBar-Auto (S.442) (Scann!!)

```
Dim cbc As CommandBarControl
  Set cbc = Application.CommandBars(”Commandbar-Auto).Controls(1). _
    Controls.Add(Type:=msoControlButton, ID:=849)
  cbc.Caption = ”Menüeintrag mit Auswahlhäkchen”
  cbc.OnAction = ”Makro1”
```

- In der Ereignisprozedur wird die Eigenschaft `.State` zwischen msoButtonDown und `–Up` umgestellt. Auf diese Weise wird das Häkchen aus- und wieder eingeblendet. Bemerkenswert an der Prozedur ist die Verwendung der Eigenschaft `ActionControl`, die auf das `CommanBarControl`-Objekt verweist, die gerade ausgewählt worden ist.

```
With CommandBars.ActionControl
  If .State = msoButtonDown Then
    .State = msoButtonUp              ‘Häkchen sichtbar
  Else
    .State = msoButtonDown            ’Häkchen unsichtbar
  End If
```

Menübefehle mit Häkchen programmieren

- Diese Art von Menübefehlen nutzen, um bestimmte Elemente in Excel ein und auszuschalten.

```
Dim i As Integer
Dim i_Hilfe As Integer
Dim MenüNeu As CommandBarControl
Dim Mb As CommandBarControl

i = Application.CommandBars(1).Controls.Count
i_Hilfe = Application.CommandBars(1).Controls(i).Index

Set MenüNeu = Application.CommandBars(1).Controls.Add _
(Type:=msoControlPopup, _
before:=i_Hilfe)
MenüNeu.Caption = ”Anal&yse”

Set Mb = MenüNeu.Controls.Add(Type:=msoControlButton)
With Mb
.Caption = ”Zellen mit Gültigkeitsregeln”
.Style = msoButtonCaption
.OnAction = ”PrüfenGültigkeit”
.State = msoButtonUp
End With
Set Mb = MenüNeu.Controls.Add(Type:=msoControlButton)
With Mb
.Caption = ”Zellen mit bedingten Formaten”
.Style = msoButtonCaption
.OnAction = ” PrüfenBedingteFormatierung ”
.State = msoButtonUp
End With
Set Mb = MenüNeu.Controls.Add _
(Type:=msoControlButton)
With Mb
.Caption = ”Zeilen ausgeblendet”
.Style = msoButtonIconAndCaption
```

```
.OnAction = ”PrüfenZeilen ”
.State = msoButtonUp
End With
Set Mb = MenüNeu.Controls.Add _
(Type:=msoControlButton)
With Mb
.Caption = ”Spalten ausgeblendet”
.Style = msoButtonIconAndCaption
.OnAction = ”PrüfenSpalten”
.State = msoButtonUp
End With
Set Mb = MenüNeu.Controls.Add _
(Type:=msoControlButton)
With Mb
.Caption = ”Tabelle geschützt”
.Style = msoButtonIconAndCaption
.OnAction = ”PrüfenSchutz ”
.State = msoButtonUp
End With
Set Mb = MenüNeu.Controls.Add _
(Type:=msoControlButton)
With Mb
.Caption = ”Hyperlinks”
.Style = msoButtonIconAndCaption
.OnAction = ”PrüfenHyperlinks”
.State = msoButtonUp
End With
End Sub
```

- Die Eigenschaft `State` gibt die Darstellung des angegebenen Schaltflächen-Steuerelements der Befehlsleiste über eine Konstante zurück oder legt sie fest. Setzen Sie bei der Erstellung die Konstante `msoButtonUp` ein, um den Kontrollhaken im ersten Schritt noch nicht anzugeigen

Die passenden Makros dazu

Existiert in der aktiven Tabelle Zellen mit Gültigkeitsregeln?

```
Sub PrüfenGültigkeit()
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
Dim i  As Integer
  i = 0
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(1)
  i = 0
  i = ActiveCell.SpecialCells(xlCellTypeAllValidation).Count
  If i > 1 Then
    ActiveCell.SpecialCells(xlCellTypeAllValidation).Select
    Mb.State = msoButtonDown
  Else
  End If
  Set Mb = Nothing
End Sub
```

Gibt es in der Tabelle Zellen mit bedingter Formatierung?

```
Sub PrüfenBedingteFormatierung()
```

```
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
Dim i As Integer
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(2)
  i = 0
  i = ActiveCell.SpecialCells(xlCellTypeAllFormatConditions).Count
 If i > 1 Then
   ActiveCell.SpecialCells(xlCellTypeAllFormatConditions).Select
   Mb.State = msoButtonDown
 Else
 End If
 Set Mb = Nothing
 Set Menü = Nothing
End Sub
```

Sind in der Tabelle Zeilen ausgeblendet?

```
Sub PrüfenZeilen()
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
Dim Zeile As Object
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(3)
  For Each Zeile In ActiveSheet.UsedRange.Rows
    If Zeile.Hidden = True Then Mb.State = msoButtonDown: Exit Sub
  Next Zeile
Set Mb = Nothing
Set Menü = Nothing
End Sub
```

Sind in der Tabelle Spalten ausgeblendet?

```
Sub PrüfenSpalten()
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
Dim Spalte As Object
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(4)
  For Each Spalte In ActiveSheet.UsedRange.Columns
    If Spalte.Hidden = True Then Mb.State = msoButtonDown: Exit Sub
  Next Spalte
  Set Mb = Nothing
  Set Menü = Nothing
End Sub
```

Ist das Tabellenblatt geschütz?

```
Sub PrüfenSchutz()
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(5)
  If ActiveSheet.ProtectContents = True Then Mb.State = ?
    msoButtonDown
  Set Mb = Nothing
  Set Menü = Nothing
End Sub
```

Gibt es auf dem Tabellenblatt Hyperlinks

```
Sub PrüfenHyperlinks()
Dim i As Integer
Dim Mb As CommandBarControl
Dim Menü As CommandBarControl
  Set Menü = CommandBars(1).Controls(”Analyse”)
  Set Mb = Menü.Controls(6)
  If ActiveSheet.Hyperlinks.Count > 0 Then Mb.State = msoButtonDown
  Menü.Execute
  Set Mb = Nothing
  Set Menü = Nothing
End Sub
```

Makro, das vorsorglich alle Hakchen zurücksetzt

- Um sicherzustellen, dass alle Haken richtig gesetzt bzw. auch wieder entfernt werden, schreiben Sie vorsorglich ein Makro, welches alle Haken vor den einzelnen Menübefehlen entfernt.

```
Sub Zurücksetzen()
Dim Mb As CommandBarControl
  Set Mb = CommandBars(1).Controls(”Analyse”)
  With Mb
    .Controls(1).State = msoButtonUp
    .Controls(2).State = msoButtonUp
    .Controls(3).State = msoButtonUp
    .Controls(4).State = msoButtonUp
    .Controls(5).State = msoButtonUp
    .Controls(6).State = msoButtonUp
 End With
End Sub
```

Workbook_SheetActivate-Ereigniss, das beim Wechsel die obigen Prüfungen ausführt

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
On Error Resume Next
  Zurücksetzen
  PrüfenGültigkeit
  PrüfenBedFormatierung
  PrüfenZeilen
  PrüfenSpalten
  PrüfenSchutz
  PrüfenHyperlinks
End Sub
```

Das Standardmenü durch ein eigenes Menü ersetzen

- Erste Experimente mit den `CommandBar`-Objekten fürhen zur Annahme, dass es seit Excel 97 (im Gegensatz zu den Vorgängerversionen) nicht mehr mögich ist, das Standardmenü durch ein eigenes Menü zu erstellen: Im interaktiven Betrieb (Symbolleisten / Anpassen) ist es nämlich unmöglich, das Standardmenü zu deaktivieren.
- Nach einigem Experimentieren stellt sich heraus: Es geht doch – allerdings nur per Programmcode: Die neue Menüleiste muss mit

Set c = CommancBars.Add(MenuBar:=True)

Erzeugt werden und weist dann `Type=msoBarTypeMenuBar` auf (und nicht wie die durch Anpassen erzeugten Symbolleisten `Type=msoBarTypeNormal`). Durch

c.Visible = True

- wird die neue Menüleiste sichtbar gemacht. Die Standardmenüleiste verschwindet automatisch (und erscheint automatisch wieder, wenn `Visible` wider auf `False` gesetzt wird). Das eigentliche Problem bei diesen Menüleisten besteht darin, dass sie (vermutlich wegen der anderslautenden `Type`-Einstellung) nicht an eine Excel-Datei angebunden werden können.
- Wenn Sie eine Excel-Programm mit einer eigenen Menüleiste ausstatten möchten, müssen Sie also alle Einträge per Code erzeugen. (Der einfachste Weg besteht sicherlich darin, eine normale Symbolleiste interaktiv zu erstellen und anzubinden. In `Wokbook_Open` kopieren Sie dann in einer Schleife alle Einträge aus dieser Smbolleiste in die durch `CommandBars.Add` erzeugte Menüleiste.) Sinnvoller ist es vermutlich, ganz auf eigene Menüleisten zu verzichten und statt dessen die anwendungsspezifischen Kommandos in einem zusätzlichen Menü in einer eigenen Symbolleiste anzuzeigen. (Das Standardmenü bleibt also weiterhin sichtbar.)
- Falls Sie sich doch dazu entschliessen, mit einem eigenen Menü zu arbeiten, sollte dieses automatisch aktivert werden, sobald ein Blatt Ihres Programms aktiv ist. Ebenso automatisch sollte aber auch das Standardmenü wieder erscheinen, sobald ein anderes Blatt oder eine andere Excel-Datei aktiviert wird. Sie können das relativ leicht erreichen, indem Sie die `Visible`-Eigenschaft der Menüleiste in `Worksheet_(De)Activate` auf `True` bzw. auf `False` setzen.

19.4 Die Programmierung von Kontextmenüs

Die gängigsten Kontextmenüs

Kontextmenü	Beschreibung
`CommandBars("Cell")`	Das Zellenkontextmenü
`CommandBars("System")`	Das System-Kontextmenü (am oberen linken Fensterrand)
`CommandBars("Toolbar List")`	Das Kontextmenü für die Menü- und Symbolleiste
`CommandBars("Ply")`	Kontextmenü für Spaltenköpfe
`CommandBars("Row")`	Kontextmenü für Zeilenköpfe

Kontextmenüs deaktivieren

```
Application.CommandBars("Cell").Enabled = True
Application.CommandBars("System").Enabled = True
Application.CommandBars("Toolbar list").Enabled = True
Application.CommandBars("Ply").Enabled = True
Application.CommandBars("Column").Enabled = True
Application.CommandBars("Row").Enabled = True
```

Kontextmenüs aktivieren

```
Application.CommandBars("Cell").Enabled = False
Application.CommandBars("System").Enabled = False
Application.CommandBars("Toolbar list").Enabled = False
Application.CommandBars("Ply").Enabled = False
Application.CommandBars("Column").Enabled = False
Application.CommandBars("Row").Enabled = False
```

Das Zellen-Kontextmenü erweitern 1

```
Dim cbb As CommandBarButton
Set cbb = Application.CommanBars("Cell").Controls.Add
cbb.Caption = "Format&vorlage"
cbb.OnAction = "Makro1"
```

Das zugehörige Makro

```
Application.Dialogs(xlDialogApplyStyle).Show
```

Zellen-Kontextmenüeintrag wieder löschen

```
Application.CommandBars("Cell").Controls("Formatvorlage").Delete
```

Das Zellenkontextmenü erweitern 2

```
Sub ZellenkontextmenüErgänzen()
Dim MB As CommandBarControl
  Set MB = Application.CommandBars("Cell").Controls.Add
  With MB
     .Caption = "Zelle reinigen"
     .OnAction = "ZellenSäubern"
     .BeginGroup = True
  End With
  Set MB = Application.CommandBars("Cell").Controls.Add
  With MB
     .Caption = "Zellenumbruch setzen"
     .OnAction = "ZellenumbruchSetzen"
```

```
  End With
  Set MB = Application.CommandBars(”Cell”).Controls.Add
  With MB
     .Caption = ”Absoluten Zellenbezug setzen”
     .OnAction = ”BezugÄndernInAbsolut”
  End With
End Sub
```

Die passenden Makros dazu

Maschinencode aus Zellen entfernen

```
Sub ZellenSäubern()
Dim Zelle As Object
  For Each Zelle In Selection
    If Zelle.HasFormula = False Then
      Zelle.Value = Application.WorksheetFunction.Clean(Zelle.Value)
    End If
  Next Zelle
End Sub
```

- Die Zellen werden mit der Funktion `Clean` gesäubert

Zeilenumbruch setzen

```
Sub ZellenumbruchSetzen()
Dim Zelle As Object
  For Each Zelle In Selection
    Zelle.WrapText = True
  Next Zelle
End Sub
```

- Setzen Sie die Eigenschaft `WrapText` auf den Wert `True`

Relative Bezüge in absolute Bezüge ändern

```
Sub BezugÄndernInAbsolut()
Dim Zelle As Object
 For Each Zelle In Selection
   If Zelle.HasFormula = True Then
    Zelle.Formula                    =                    Applica-
tion.ConvertFormula(Formula:=Zelle.Formula, _
     fromreferencestyle:=xlA1,    toreferencestyle:=xlA1,    toabso-
lute:=xlAbsolute)
  Else
  End If
 Next Zelle
End Sub
```

- Die Methode `ConvertFormula` konvertiert Zellenbezüge in Formeln zwischen den Bezugsarten A1 und Z1S1 bzw. zwischen relativen und absoluten Bezügen.

Kontextmenü zurücksetzen

```
Application.CommandBars(”Cell”).Reset
```

Die vorher eingefügten Befehle einzeln entfernen

```
Sub ZellenkontextmenüBefehleLöschen()
Dim Menü As CommandBar
  Set Menü = Application.CommandBars(”Cell”)
  On Error Resume Next
  With Menü
```

```
    .Controls(”Zelle reinigen”).Delete
    .Controls(”Zellenumbruch setzen”).Delete
    .Controls(”Absoluten Zellenbezug setzen”).Delete
  End With
End Sub
```

Eigene Kontextmenüs

- Innerhalb von Tabellenblätter und Diagrammen (und nur dort) können Sie die `BeforeRightClick`-Ereignisprozedur dazu ausnutzen, um die automatische Anzeige diverser Excel-Kontextmenüs (abhängig davon, welche Objekte gerade markiert sind) zu verhindern und statt dessen ein eigenes Kontextmenü anzeigen. Die erste Voraussetzung besteht darin, dass Sie vorher ein entsprechendes Menü mit `Postion=msoBarPopup` definieren:

```
Private Sub Workbook_Open()
Dim cb As CommandBar
  Set cb = Application.CommandBars.Add(Name:=”NewPopup”, Posi-
tion:=msoBarPopup)
  With cb
     .Controls.Add Type:=msoControlButton, Id:=3  'Speichern
     .Controls.Add …
  End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
  Application.CommandBars(”NewPopup”).Delete
End Sub
```

- Der Aufruf des Menüs kann prinzipiel an einer beliebigen Stelle im Code mit der Methode `ShowPopup` erfolgen. Für das vorliegende Beispiel wurde die `BeforeRightClick`-Ereignisprozedur des zweiten Tabellenblatts gewählt. Durch die Zuweisung `Cancel=True` wird erreicht, dass anschliessend nicht auch noch Excel ein Kontextmenü anzeigt.

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As _
Excel.Range, Cancel As Boolean)
  Application.CommandBars(”NewPopup”).ShowPopup
  Cancel = True
End Sub
```

19.5 Die Programmierung von Symbolleisten

- Im Regelfall ist es zwar nicht sinnvoll, ganze Symbolleisten per Programmcode zu erzeugen – das geht viel einfacher im manuellen Betrieb. Oft besteht aber die Notwendigkeit, je nach Zustand des Programms Symbolleisten oder auch nur einzelne Menüeinträge ein- oder auszublenden, deren Text zu verändern etc. Excel-Anwendungen mit einer eigenen Symbolleiste sollten sich auch darum kümmern, dass diese beim Laden automatisch angezeigt und beim Schliessen wieder entfernt wird.

Symbolleiste ein- und ausblenden

```
Application.CommandBars(”Neue Symbolleiste”).Visible = True
```

- Beim Programmende kann die Symbolleiste druch `Visible=False` wieder unsichtbar gemacht werden. Die Symbolleiste bleibt dann allderdings im Speicher und wird in der Excel.xlb-Datei gespeichert. Damit die Anzahl der so gespeicherten Symbolleisten nicht unbegrenzt ansteigt, ist es sinnvoll, die Symbolleiste explizit mit `Delete` zu löschen. `Delete` bezieht sich dabei auf Excel. Innerhalb der aktiven Excel-Datei bleibt die angebundene Symbolleiste erhalten

```
Application.CommandBars(”Neue Symbolleiste”).Visible = False
Application.CommandBars(”Neue Symbolleiste”).Delete
```

Neue Symbolleist erstellen

```
Dim SB As CommandBar
  On Error Resume Next
  Set SB = CommandBars.Add(”Neue Symbolleiste”)
  With SB
    .Visible = True
    .Top = 400
    .Left = 70
  End With
```

- Mit Hilfe der Methode `Add` fügen Sie eine neue Symbolleiste ein. Dabei bestimmen Sie über die Eigenschaft Visible, dass die Symbolleiste auf dem Bildschirm angezeigt wird. Mit den Eigenschaften `Top` und `Left` legen Sie die exakte Anzeigeposition (linke obere Ecke) fest.

Makro, dass ID-Nummern und Symbolnamen ermittelt

- Jede einzelne Symbolschaltfläche in Excel hat eine eindeutige `ID` und einen Namen, worüber Sie die Symbolschaltfläche ansprechen können.

```
Sub ID()
Dim SBS As Object
Dim SB As CommandBar
Dim SBS
Dim i As Long
On Error Resume Next
For i = 1 To 32000
  Set SBS = CommandBars.FindControl _
  (Type:=msoControlButton, ID:=i)
  If Range(”A1”).Value = ”” Then
    Range(”A1”).Value = i & ” ” & SBS.Caption
  ElseIf Range(”A2”).Value = ”” Then
```

```
    Range(”A2”).Value = i & ” ” & SBS.Caption
  Else
    Range(”A1”).Select
    Selection.End(xlDown).Select
    ActiveCell.Offset(1, 0).Select
    ActiveCell.Value = i & ” ” & SBS.Caption
  End If
Next i
```

- Über die Methode `FindControl` können Sie nach der Symbolschaltfläche suchen, deren Index die For Schleife hat. Wird ein Symbol gefunden, können Sie die Beschriftung und die Indexnummer der Symbolschlatfläche in der Spalte A ausgeben.

Symbolschaltfläche in die Symbolleiste einfügen

```
Dim SL As CommandBar
  Set SL= CommandBars(”Neue Symbolleiste”)
  Set Symbol = SL.Controls
  Symbol.Add Type:=msoControlButton, ID:=3
```

Löschen von Symbolen mit der Methode FindControl

```
Dim SLS As Object
Dim SL As CommandBar
  Set SL = CommandBars(”Neue Symbolleiste”)
  Set SLS = SL.FindControl(Type:=msoControlButton, ID:=Text1.Value)
  SBS.Delete
```

- Übergeben Sie der Methode `FindControl` den ID-Wert der zu löschenden Symbolschaltfläche. Wird die gesuchte Symbolschaltfläche auf der neuen Symbolleiste gefunden, wenden Sie die Methode `Delete` an, um sie zu löschen.

Grafik in Symolleiste integrieren und ein Makro zuweisen

1. Fügen Sie die Grafik in die Tabelle ein
2. Klicke Sie mit der rechten Maustaste auf die Grafik und notieren Sie sich den angezeigten Namen der Grafik, den Sie aus dem Namensfeld ersehen können.
3. Starten Sie folgendes Makro:

```
Dim Symbol As CommandBarButton
Dim SBS As Object
  Set Symbol = CommandBars(”Standard”).Controls. _
    Add(msoControlButton)
  Worksheets(”Tabelle1”).Shapes(”AutoForm 7”).CopyPicture
  Symbol.PasteFace
  Set SBS = Symbol
  SBS.OnAction = ”Animation”
```

- Fügen Sie der Symbolschaltfläche Standard eine neue Symbolschaltfläche zu, die Sie aus der eingefügten AutoForm Smiley aus der Tabelle1 mit Hilfe der Methode `CopyPicture` kopieren und mit `PasteFace` einfügen. Um dieser neuen Symbolschaltfläche ein Makro zuzuweisen, setzen Sie die Eigenschaft `OnAction` ein.

Symbolschaltflächen deaktivieren

```
Dim SB As CommandBar
Dim i As Integer
  Set SB = CommandBars(”Standard”)
  With SB
```

```
  i = .Controls.Count
  .Controls(i).Enabled = False
 End With
End Sub
```

Symbolschaltflächen aktivieren

```
Dim SB As CommandBar
Dim i As Integer
  Set SB = CommandBars(”Standard”)
  With SB
  i = .Controls.Count
  .Controls(i).Enabled = True
 End With
End Sub
```

Dropdown-Element in Symbolleiste einfügen (Einträge mit Symbol und Text)

```
Dim SB As CommandBar
Dim DropSym As CommandBarControl
Dim Symbol As Object
Dim NeuSymb As Object

  On Error Resume Next
  Set SB = CommandBars.Add(”Dropdown”)
  SB.Visible = True
  Set DropSym = SB.Controls.Add(Type:=msoControlPopup)
  DropSym.Caption = ”Einfügen”

    Set DropSym = SB.FindControl(Type:=msoControlPopup)
    Set Symbol = DropSym.Control.CommandBar

    Set NeuSymb = Symbol.Controls.Add _
     (Type:=msoControlButton, ID:=1576)
    NeuSymb.Caption = ”Hyperlink”

    Set NeuSymb = Symbol.Controls.Add _
     (Type:=msoControlButton, ID:=385)
    NeuSymb.Caption = ”Funktion”

    Set NeuSymb = Symbol.Controls.Add _
     (Type:=msoControlButton, ID:=436)
    NeuSymb.Caption = ”Diagramm”

    Set NeuSymb = Symbol.Controls.Add _
     (Type:=msoControlButton, ID:=1589)
    NeuSymb.Caption = ”Kommentar”
```

Dropdown-Element mit Text in Symbolleiste einfügen

```
Dim SB As CommandBar
Dim Symbol As Object
Dim i As Integer
  Set SB = CommandBars.Add(”Auswahl”)
  SB.Visible = True
  Set Symbol = Application.CommandBars(”Auswahl”).Controls. _
```

```
    Add(Type:=msoControlComboBox)
  With Symbol
     Sheets(”Tabelle3”).Activate
     Range(”H1”).Select
     i = 1
     Do Until ActiveCell.Value = ””
      .AddItem Text:=ActiveCell.Value, index:=i
      ActiveCell.Offset(1, 0).Select
      i = i + 1
     Loop
    .DropDownLines = 4
    .DropDownWidth = 100
    .ListHeaderCount = 0
    .OnAction = ”Auswahl”
End With
```

- Geben Sie der Methode Add den richtigen Typ der Symbolschaltfläche an. Da Sie ein Dropdown mit Textwerten haben möchten, müssen Sie daher die Konstante `msoControlComboBox` verwenden. Danach lesen Sie die einzelnen Inhalte des Dropdown-Elements aus der Tabelle3 ein. Über die Eigenschaft `DropDownLines` geben Sie an, wie viele Listenzeilen im Dropdown-Element angezeit werden sollen. Mit der Eigenschaft `DropDownWidth` können Sie die Breite des Dropdown-Elements in Pixel angeben. Die Eigenschaft `ListHeaderCount` können Sie einsetzen, wenn Sie einen horizontalen Trennstrick im Dropdown-Elemen haben und die Anzahl der Listenelemente bestimmen möchten, die über der Trennlinie angezeigt werden. Dabei bedeutet der Wert 0, dass kein Listenelement über der Trennlinie angezeigt werden soll. Die Eigenschaft `OnAction` verweist auf ein Makro, welches ausgeführt werden soll, wenn ein Eintrag aus dem Dropdown-Element ausgewählt wird.

- Nun müssen Sie nur noch ermitteln, welcher Eintrag im Dropdown-Element ausgewählt wurde. Dafür ist das folgende Makro verantwortlich

```
Sub Auswahl()
Dim index As Integer
  index                              =                              Applica-
tion.CommandBars(”Auswahl”).Controls.Item(1).ListIndex
  Select Case index
  Case 1
     Call Makro1
  Case 2
     Call Makro2
  Case 3
     Call Makro3
  Case 4
     Call Makro4
  Case Else
      MsgBox ”Noch nicht zugeordet!”
 End Select
```

- Mit Hilfe der Eigenschaft `ListIndex` bekommen Sie den momentan markieten Eintrag des Dropdown-Elements zurück. Diesen werten Sie dann in einer `Select Case`-Anweisung aus und starten danach das jeweils zugeordnete Makro.

Blattwechsel per Symbolleiste mit Listenfeld

- Die hier vorgestellt Symbolleiste enthält ein Listenfeld (`CommandBarComboBox`), das diese Aufgabe erleichtert. Die eigentliche Herausforderung bei der Programmierung ist die Synchronisation des Listenfelds mit der Blattliste der gerade aktiven Arbeitsmappe. Dazu müssen zwei Ereignisse des `Application`-Objekts ausgewertet werden, was ein eigenes Klassenmodul voraussetzt

Listenfeld erstellen

- Das Listenfeld kann nur per Programmcode (nicht interaktiv) erzeugt werden. Wenn die Datei zum zweiten Mal von derselben Person geldaden wird, kann allerdings sein, dass das Listenfeld noch in der Symbolleiste existiert. Aus diesem Grund befindet sich in `Workbook_Open` eine diesbezügliche Sicherheitsabfrage.
- Das Listenfeld wird aus Platzgründen nicht beschriftet (also kein `Caption`-Text). Dafür wird eine Zeichenkette an `TooltipText` zugewiesen, die als gelber Infotext angezeigt wird. Ausserdem wird die *Tag*-Eigenschaft belegt, die an anderen Stellen im Code den Einsatz der Methode `FindControl` ermöglicht (siehe unten). Die Einstellung der beiden Eigenschaften `DropDownLines` und `DropDownWidth` bewirkt, dass das Listenfeld etwas grösser als in der Defaulteinstellung erscheint. (Das ist bei langen Listen übersichtlicher.)

```
Dim appc As New AppClass          'Für die Applications-Ereignisse

Private Sub Workbook_Open()
Dim cbc As CommandBarControl, cbcb As CommandBarComboBox
Dim existing As Boolean
  On Error Resume Next
  'testen, ob das Listenfeld in der Symbolleiste »Neue Symbolleiste«
  schon existiert
  For Each cbc In Application.CommandBars _
    (”Neue Symbolleiste”).Controls
    If cbc.Tag = ”Blattliste” Then existing = True: Exit For
  Next
  'Listenfeld in die Symbolleiste »Neue Symbolleiste« eintragen
  If Not existing Then
    Set cbcb = Application.CommandBars(”NeueSymbolleiste”). _
      Controls.Add(Type:=msoControlDropdown, Before:=2)
    cbcb.Tag = ”Blattliste”
    cbcb.TooltipText = ”Blattliste”
    cbcb.OnAction = ”Makro1”
    cbcb.DropDownWidth = 150
    cbcb.DropDownLines = 20
  End If
'Symbolleiste »Neue Symbolleiste« anzeigen
Application.CommandBars(”Neue Symbolleiste”).Visible = True
'Ereignis für Arbeitsmappenwechsel empfangen
Set appc.app = Application
End Sub
```

Listenfeld mit Einträgen belegen

- Vielleicht haben sie in den Zeilen oben Anweisungen vermisst, mit denen die Listeneinträge mit Blattname belegt werden. Da sich die Blattliste ständig ändert (beim Wechsel zwischen zwei Arbeitsmappen, beim Laden neuer Dateien, beim Einfügen und Löschen von Blättern), muss das Listenfeld immer wieder neu mit Einträgen belegt

werden. Der erforderliche Code befindet sich im Klassenmodul *AppClass*, das in den oben abgedruckten Zeilen initialisiert wurde (siehe die *Dim*-Zeile bzw. die letze Zeile).

- Die Ereignisprozedur *app_SheetActivate* wird immer dann aufgerufen, wenn ein Blattwechel in irgendeiner Excel geladenen Datei stattfindet. In der Prozedur wird das Listenfeld zuerst mit *FindControl* gesucht. Anschliessend wird der aktuelle Inhalt der Liste gelöscht und durch die Namen der Blätter der aktiven Arbeitsmappe ersetzt. Schliesslich wird jener Eintrag der Liste aktiviert, der dem momentan aktiven Blatt entspricht.

- *App_WorkbookActivate* ruft einfach *app_SheetActivate* auf, um auch bei einem Wechsel der Arbeitsmappe das Listenfeld zu aktualisieren

Im Klassenmodul »AppClass«

```
' es sollen Application-Ereignisse verarbeitet werden

Public WithEvents app As Application

'Blattliste neu erstellen (das Activate-Ereignis tritt auch auf,
wenn die Arbeitsmappe  gewechselt, ein neues Blatt eingefügt oder
das aktuelle Blatt gelöscht wird) sh.Parent liefert ein Workbook-
Objekt

Private Sub app_SheetActivate(ByVal sh As Object)
Dim cbcb As CommandBarComboBox
Dim sheet As Object
Dim i
  Set cbcb = Application.CommandBars.FindControl _
   (Type:=msoControlDropdown, Tag:=”Blattliste”)
  cbcb.Clear
  For Each sheet In sh.Parent.Sheets
    If TypeName(sheet) <> ”Module” Then
      cbcb.AddItem sheet.Name
    End If
  Next
  ' richtigen Listeneintrag aktivieren
  For i = 1 To cbcb.ListCount
    If cbcb.List(i) = sh.Name Then cbcb.ListIndex = i: Exit For
  Next
End Sub

Private Sub app_WorkbookActivate(ByVal wb As Excel.Workbook)
  app_SheetActivate wb.ActiveSheet
End Sub
```

Symbolleiste gegen Veränderung schützen

- Mit der `Protection`-Eigenschaft des `CommandBar`-Objekts kann stufenweise eingestellt werden, welche Veränderung an der Symbolleiste möglich sind: gar keine, Position ändern, Grössen ändern, Inhalt ändern etc. Durch eine entsprechende Einstellung können Sie also Manipulationen durch den Anwender vermeiden

19.6 Syntaxzusammenfassung

CommandBars – Methoden und Eigenschaften

ActiveMenuBar	verweist auf *CommandBar*-Objekt mit aktiver Menüleiste
Add	neue Symbolleiste hinzufügen
FindControls	Element in Symbolleisten suchen

CommandBar – Methoden und Eigenschaften

ActionControl	verweist auf das gerade angeklickte Symbol / Menüelement
BuiltIn	*True* bei vordefinierten Symbolleisten
Controls	Zugriff auf Symbole bzw. Menüeinträge
Delete	Symbolleiste löschen
Name	englischer Name der Symbolleiste
NameLocal	Name in der jeweiligen Landesprache (Deutsch)
Position	Ort (verankert oder als Toolbox)
Protection	Schutz vor Veränderungen durch den Anwender
ShowPopup	als Kontextmenü anzeigen
Visible	Sichtbarkeit der Symbolleiste

CommandBarControls – Methoden und Eigenschaften

Add	Symbol / Menüeintrag / Liste hinzufügen
Count	Anzahl der Menüelemente bzw. Symbole

CommandBarControl – Methoden und Eigenschaften

BeginGroup	mit dem Objekt beginnt eine Gruppe
BuiltIn	*True* bei vordefinierten Elementen
Caption	Bechriftungstext
Copy	kopiert einen Eintrag aus einer anderen Symbolleiste
Delete	Eintrag löschen
Enabled	*True,* wenn das Element verwendet warden kann
Execute	führt die *OnAction*-Prozedur aus
OnAction	Name der Ereignisprozedur
TooltipText	gelber Infotext (bei *TooltipText=””* wird *Caption* verwendet)
Type	Typ (z.B. *msoControlButton, -ComboBox, -Popup*)

CommandBarButton

BuiltInFace	*True,* falls vordefiniertes Symbol
CopyFace	Symbol in die Zwischenablage kopieren
PasteFace	Symbol aus der Zwischenablage einfügen
Reset	Eintrag zurücksetzen (nur sinnvoll wenn *BuiltIn = True*)

CommandBarComboBox – Methoden und Eigenschaften

AddItem	Listenelement hinzufügen
Clear	alle Listenelemente löschen
DropDownLines	gewünschte Anzahl der Zeilen für ausklappbare Liste
DropDownWidth	gewünschte Breite der ausklappbaren Liste

List(n)	Zugriff auf Listenelemente
ListCount	Anzahl der Listenelemente
ListIndex	Indexnummer des ausgewählten Elements
RemoveItem	Listenelement löschen
SetFocus	Eingabefokus auf das Listenfeld richten
Text	eingegebener Text bzw. Text des ausgewählten Elements
Type	*msoControlEdit*, *-Dropdown* oder *-ComboBox*

CommandBarPopup – Methoden und Eigenschaften

Controls	Zugriff auf Element (verweist auf*CommandBarControls*)

Bei den Objekten `CommandBarButton`, `CommandBarComboBox` und `CommandBarPopup` wurden nur jene Methoden und Eigenschaften aufgezählt, die nicht ohnedies bereits von `CommandBarControl` zur Verfügung gestellt werden.

20 Excel und das Mailing

Aktives Tabellenblatt als E-Mail versenden

```
Sub EmailAktivesTabellenblatt()
Dim s As String
Dim s2 As String
 ActiveWorkbook.Worksheets(ActiveSheet.Name).Copy
 s = InputBox(”Bitte geben Sie den Adressaten ein!”, ”Adressat”)
 If s = ”” Then MsgBox ”Sie haben abgebrochen!”: Exit Sub
 s2 = InputBox(”Bitte geben Sie den Titel des E-Mails ein!”, _
   ”Titel”)
 If s2 = ”” Then MsgBox ”Sie haben abgebrochen!”: Exit Sub
 Application.Dialogs(xlDialogSendMail).Show s, s2
 ActiveWorkbook.Close savechanges:=False
End Sub
```

Tabelle als E-Mail versenden

```
Dim Empfänger As String
  Empfänger = InputBox(”Geben Sie den Empfänger des e-Mails ein!”)
  If Empfänger = ”” Then Exit Sub
  ActiveWorkbook.ActiveSheet.Copy
  ActiveWorkbook.SaveAs Range(”A1”).Value & ”.xls”
  ActiveWorkbook.SendMail Recipients:=Empfänger,
    Subject:=Range(”A1”).Value
  ActiveWorkbook.Close savechanges:=False
```

- Im Argument `Recipients` geben Sie den E-Mail-Empfänger an, den Sie vorher über die Funktion `Inputbox` vom Anwender erfragt haben. Im Argument `Subject` geben Sie den Namen des Betreffs ein, den Sie ebenfalls aus der Zelle A1 ermitteln. Wenn Sie möchten, können Sie noch das Argument `ReturnReceipt` einsetzen, das Sie auf den Wert `True`, setzen, sofern Sie eine Empfangsbestätigung möchten.

Excelbereich als E-Mail versenden

```
Dim Empfänger As String
Dim Bereich As Range
  Application.SheetsInNewWorkbook = 1
  Empfänger = InputBox(”Geben Sie den Empfänger des e-Mails ein!”)
  If Empfänger = ”” Then Exit Sub
  Set Bereich = Application.InputBox _
    (”Wählen Sie den Bereich aus den Sie versenden möchten”, _
    Type:=8)
  Range(Bereich.Address).Select
  Selection.Copy
  Workbooks.Add
  ActiveSheet.Paste
  ActiveWorkbook.SaveAs ”Anhang.xls”
  Application.Dialogs(xlDialogSendMail).Show Empfänger, ”markierter
Bereich”
End Sub
```

Mehrere Arbeitsmappen per E-Mail versenden

- Möchten Sie regelmässig eine ganze Reihe von Dateien versenden, können Sie diesen Vorgang automatisieren. Im folgenden Beispiel werden alle Dateien aus dem Verzeichnis C:\RECHNUNGEN\ als E-Mail-Anhang in Outlook versendet.

```
Dim outObj As Object
Dim Mail As Object
Dim i As Integer
  Set outObj = CreateObject("Outlook.Application")
  Set Mail = outObj.CreateItem(0)
  With Mail
    .Subject = "Rechnungen"
    .Body = "Sehr geehrte Damen und Herren " & Chr(13) & _
    "Bitte prüfen Sie die angehängten Rechnungen" & Chr(13) & _
      "Viele Grüße " & Chr(13) & _
      Application.UserName
    .To = "Rewe@Mac.de"
    .CC = "Fibu@Mac.de"
   End With
      With Application.FileSearch
        .NewSearch
        .LookIn = "c:\Rechnungen\"
        .SearchSubFolders = True
        .FileType = msoFileTypeAllFiles
        .Execute
        For i = 1 To .FoundFiles.Count
           Mail.Attachments.Add .FoundFiles(i)
        Next i
      End With
   Mail.Display
   Set Mail = Nothing
   Set outObj = Nothing
  End Sub
```

- Definieren Sie zuerst zwei Objektvariablen. Die erste Variable "outObj" soll einen Verweis auf die Anwendung Outlook darstellen, die zweite Variabel "Mail" gibt einen Verweis auf die Outlook-Komponente, die für das Erstellen und Verschicken von E-Mails verantwortlich sit. Über die Funktion `CreateObject` erstellen Sie ein Outlook-Objekt. Mit der Methode `CrateItem` erstellen Sie ein Outlook-Element. Entnehmen Sie der folgenden Tabelle die einzelnen Elemente, die Sie in Outlook erstellen können.

Index	Konstante	Aufgabe in Outlook
0	olMailItem	Erstellen von E-Mails
1	olAppointmentItem	Termine bearbeiten und verwalten
2	olContactItem	Bearbeiten und Verwalten von Kontakten
3	olTaskItem	Bearbeiten und Verwalten von Aufträgen
4	olJournalItem	Journaleinträge erstellen und verwalten
5	olNoteItem	Bearbeiten und Verwalten von Notizen
6	olPostItem	Verschicken von E-Mails

- Wenn Sie die Online-Hilfe nützen möchten, um zu den einzelnen Outlook-Befehlen mehr Informationen zu bekommen, müssen Sie die Objektbibliothek MICORSOFT OUTLOOK 9.0 OBJECT LIBRARY unter dem Menü Extras / Verweise einbinden.

- Legen Sie mit der Eigenschaft `Subject` den Betreff der E-Mail-Nachricht an. Über die Eigenschaft `Body` können Sie den E-Mail-Text festlegen. Mit dem Zeichen & können Sie diesen Text auch mehrzeilig schreiben. Mit Hilfe der Eigenschaft `To` geben Sie die E-Mail-Adresse des Empfängers der E-Mail an. Möchten Sie die E-Mail an mehrere Empfänger schicken, dann geben Sie die einzelnen E-Mail-Adressen durch ein Semikolon getrennt ein. Wenn Sie die E-Mail als Kopie an weitere Empfänger senden möchten, die nicht unmittelbar davon betroffen sind, dann verwenden Sie die Eigenschaft ***CC***.
- Setzen Sie `FileSearch` ein, um alle Dateien im Verzeichnis C:\RECHNUNGEN zu ermitteln, und hängen Sie diese mit Hilfe der Methode `Add` als Anhang an Ihre E-Mail an. Zeigen Sie zum Abschluss den E-Mail-Dialog über die Methode `Display` an und geben Sie den reservierten Speicherplatz für die Objektvariable wieder frei.

E-Mail bei Änderung senden

Innerhalb der Tabelle soll nur eine bestimmte Spalte überwacht werden.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
Dim outObj As Object
Dim Mail As Object

 If Target.Column = 2 Then
  ActiveWorkbook.Save
  Set outObj = CreateObject("Outlook.Application")
  Set Mail = outObj.CreateItem(0)
  Mail.Subject = "Änderungen"
  Mail.Body = "Hallo Kollegen, " & Chr(13) & _
      "Bitte die Arbeitsmappe Umfeld.xls/Tabelle6 überprüfen." & _
      Chr(13) & " Diese Arbeitsmappe wurde geändert" & Chr(13) & _
      "M.f.G" & Chr(13) & Application.UserName
  Mail.To = "Philippin@Mac.de; Vogelgsang@Mac.de"
  Mail.CC = "Machero@aol.com"
  Mail.Display
 End If
 Set Mail = Nothing
 Set outObj = Nothing
End Sub
```

21 Eigendefinierte Datentypen

- Wirklich neue Datentypen können Sie nicht konstruieren, aber immerhin Typen, die sich aus beliebigen Kombinationen der Grundtypen zusammensetzen – zumindest fast beliebigen Kombinationen, denn bei Strings dürfen nur Strings fester Länge verwendet werden.
- Eigene Datentypen, sogenannte ”Verbunde”, ”Strukturen” oder auch ”Records”, die sich aus mehreren Variablen zusammensetzen, definieren Sie mit der `Type`-Anweisung:

Die Typeanweisung

```
Type Benutzertyp
  Elementname As Typname
  Elementname As Typname
  …
  …
End Typ

Type Adresstype
  Vorname As String * 20
  Nachname As Sting * 30
  Plz As Long
  Wohnort As String
End Type
```

- Die Deklaration eigendefinierter Datentypen ist nur ”auf Modulebene” möglich. Im Abschnitt: ”(Deklaration)”. Keinesfalls darf eine solche Deklaration innerhalb einer Prozedur erfolgen.
- Nach der Deklaration können alle Prozeduren dieses Moduls auf den neuen ”privaten” Datentyp des Moduls zugreifen. Wenn sie wollen, können Sie das Schlüsselwort `Private` verwenden, um diesen Sachverhalt zu verdeutlichen.

```
Private Type Benutzertype
```

- Mit dem Zusatz `Public` deklarieren Sie dagegen Datentypen, auf die anschliessend alle Prozeduren aller Module zugreifen können:

```
Public Type Benutzertype
```

Definition von Variablen

- Beachten Sie, dass der neue Datentyp `Adresstyp` bisher nur *deklariert* wurde, also bekanntgegeben, aber dass noch keine Variablen *definiert*, das heisst angelegt wurden
- Benötigen Sie in irgendeiner Prozedur Ihres Programms eine Variable des neuen Typs `Addresstyp`, müssen Sie sie zuvor mit `Dim` und `As` definieren. Z.B. definiert:

```
Dim Adr As Adresstyp
```

- eine Variable namens `Adr`, die weder eine numerische noch eine Stringvariable ist, sondern eine Variable vom Typ `Adresstyp`.
- `Adr` ist eine ”zusammengesetzte” Variable, ein Verbund mehrerer Variablen. Zum Sprachgebrauch: Adr ist eine ”**Verbundvariable**” oder ”**Recordvariable**”, die aus einzelnen Variablen besteht, den ”Komponenten” dieses Datentyps.

- Ein Verbund besitzt eine gewisse Ähnlichkeit mit einem Array, das ja ebenfalls mehrere Elemente enthält. Allerdings kann ein Verbund im Gegensatz zu einem Array Komponenten unterschiedlichsten Typs enthalten.

```
Variablenname.Komponente
Adr.Vorname = ”Wille
```

Normal

```
Adr.Vorname = ”Willi”
Adr.Nachname = ”Maier”
Adr.Plz = 23000
Adr.Wohnort = ”Mannheim”
```

Mit With

```
With Adr
  .Vorname = ”Willi”
  .Nachname = ”Maier”
  .Plz = 23000
  .Wohnort = ”Mannheim”
End With
```

Zuweisung

- Sie können eine Verbund- oder Recordvariable einer anderen Recordvariable zuweisen, die *vom gleichen Typ* ist.

```
Dim Adr1 As Adresstyp, Adr2 As Adresstyp
Adr2 = Adr1
```

- Natürlich können Sie auch einzelne Komponenten in die zweite Adressvariable kopieren

```
Adr2.Plz = Adr1.Plz
```

Übergabe

- Eine Verbundvariable können Sie komplett an eine Prozedur/Funktion übergeben, allerdings nur als Referenz, nicht als Wert.

```
Call Demo(Adr)
```

- Die Parameterliste von Demo muss eine Übernahmevariable vom gleichen Typ aufweisen

```
Sub Demo(Adresse As Adresstyp)
```

- Ähnlich übergeben Sie eine einzelne Komponente

```
Call Demo(Adr.Vorname)
```

- Die zugehörige Prozedur-Parameterliste muss den gleichen gleichen Typ (String * 20) wie die übergebene Komponente besitzen

```
Sub Demo(s As String * 20)
```

- Sie können auch Arrays aus Verbundvariablen deklarieren. Z.B. so:

```
Type Adresstype
 Vorname As String * 20
 Nachname As Sting * 30
 Plz As Long
 Wohnort As String
End Type

Public Sub Typ_Test()
 Dim adr(1 To 10) As Adresstyp
 adr(1).Nachname = "Maier"
 adr(2).Nachname = "Müller"
End Sub
```

- Das Array *adr* besteht aus den zehn Variablen `adr(1)` bis `adr(10)`, die jeweils vom Typ `Adresstyp` sind und denen wie in der Abbildung gezeigt nun zum Beispiel komponentenweise Daten zugewiesen werden können. Für den Zugriff gilt somit die Syntax:

```
Datenfeldname(Index).Komponente
```

Verschachtelte Verbunde

- Jede Komponente eines Verbunds darf nicht nur aus einer einfachen Variablen bestehen, sondern selbst aus einem Verbund:

```
'Innerer Verbund
Type AdressOrt
  Plz As Long
  Ort As String * 40
End Type

'Äusserer Verbund
Type Adresstype
  Vorname As String * 20
  Nachname As String * 30
  Wohnort As AdressOrt
End Type

'Verbundvariable
Dim Adr As Adresstyp
```

Der Verbund `Adresstyp` besteht aus zwei einfachen Komponenten und zusätzlich aus einer Komponenten `Wohnort`, die selbst einen Verbund mit zwei Komponenten darstellt.

- Die Frage ist, wie eine Komponente dieses "inneren" Verbunds angesprochen wird, z.B. die Plz:

```
Variablenname.KomponenteA.KomponenteI
```

- KomponenteA ist eine Komponente des "äusseren" Verbunds und "KomponenteI" eine Komponente des inneren Verbunds. Entsrpechend ist

```
Adr.Wohnort.Plz
```

- ein Bezug auf die in diesem inneren Verbund enthaltene Komponente Plz. Verbunde können beliebig tief ineinander verschachtelt sein. Bezüge auf ”tief im Innern” enthaltene Komponenten besitzen folgende Form:

```
Variablenname.Komponente.Komponente.Komponente…..
```

22 Felder

Arrays

- Array = Datenfeld. Ist eine Gruppe von Variablen gleichen Datentyps
- `Dim Arrayname(IndexUntergrenze To IndexObergrenze) As Datentyp`
- `Dim Arrayname(IndexObergrenze) As Datentyp`
- z.B. `Dim Wochentag(1 To 7)`
- `Array` können Sie mit `Dim`, `Static`, `Private`, `Public` oder `ReDim` deklarieren

Dynamische Arrays deklarieren

- `Dim Arrayname() As Datentyp`
- Es wird keine untere oder obere Indexgrenze angegeben

Grösse dynamischer Array verändern

- `ReDim` Arrayname(Dimension1, Dimension2, …)
- `ReDim Preserve` Arrayname(Dimension1, Dimension2)

Wert an Feld übergeben

- Varablenname(Index)

- `x(5) = 100`
  Weist der Variablen Nummer 5 des Array x() den Wert 100 zu.

- `zahl = 5`
  `x(zahl) = 100`
  Zahl enthält eine `5`, also wird in der folgenden Zuweisung das Element Nummer `5` des Array `x()` angesprochen, dass heisst die Variable `x(5)`.

Lösung ohne Array

```
Sub Test
Dim x1 AS Integer
…
Dim x100 As Integer

  x1 = 1
  …
  x100 = 100
End Sub
```

Lösung mit Array

```
Dim x(1 To 100)
Dim i As Integer
  For i = 1 To 100
    x(i) = i
    Debug.Print x(i)
  Next i
End Sub
```

Deklarieren eines Arrays mit zehn Single-Variablen

- `Dim x(10) As Single`
  `x(0), …, x(9)`

Option Base

- Die Array-Untergrenze 0 können Sie mit der im Deklarationsabschnitt des Moduls eingefügten Anweisung `Option Base {0/1}` verändern.

To

- Es ist aussagekräftiger, bei der Deklaration das Schlüsselwort `To` einzusetzen `Dim x(5 To 8) As Single`

Array an Prozedur übergeben

- `Call Prozedurname(Arrayname(Index))`
  `Call Test(x(3),x((8)))`
- Prozedur übergibt `x(8)` nicht als Referenz, sondern als Wert. Die Parameterliste von Test muss wie immer eine korrespondierende Variable vom gleichen Typ enthalten. Z.B. zwei `Single`-Variablen, wenn das Array `x()` Variablen dieses Typs enthält.

- `Sub Test(Zahl1 As Single, Zahl2 As Single)`
- Jede Änderung von `Zahl1` beeinflusst die als Referenz übergebene zugehörige Arrayvariable `x(3)`, Zuweisungen an Zahl2 können den Wert der "als Wert" übergebenen Variablen `x(8)` dagegen nicht verändern.

- Komplette Arrays können Sie nur als Referenz übergeben. Der Ausdruck
  `Call Test(x())`
  übergibt "Test" das gesamte Array. Die Argumentenliste muss ein entsprechendes "Übernamearray" gleichen Typs enthalten:

- `Sub Test(zahl() As Single)`
- Die grösse des Übernahmearrays wird nicht angegeben, sondern nur eine Variable mit leeren Klammern als Kennzeichen dafür, dass es sich um ein Array handelt. Test besitzt nun Zugriff auf jedes einzelne Arrayelement und kann zum Beispiel mit einer Zuweisung wie `Zahl(5) = 23.4` den Inhalt der korrespondierenden Variable `x(5)` verändern.

UBound und LBound

- Bei der Übergabe von Arrays an Prozeduren spielen die Arraygrenzfunktionen `UBound` und `LBound` eine wichtige Rolle. Sie ermitteln den grössten (UBound) bzw. kleinsten (LBound) Index eines Array und übergeben ihn als Funktionswert:

- `UBound(Arrayname)`
- `LBound(Arrayname)`

- Wurde ein Array mit `Dim Zahl(5 To 23) As Single` deklariert, übergibt `UBound(x)` den Wert `5` und `LBound(x)` den Wert `23`.
- `UBound` und `LBound` ermöglichen es, sehr flexible Prozeduren zu schreiben, denen ein Array unbekannter Grösse übergeben wird und die dennoch alle Elemente dieses Arrays manipulieren, indem in der Prozedur selbst mit `UBound` und `LBound` die Arraygrenzen ermittelt werden:

```
Sub Test(zahl() As Single)
  Dim i As Single
  For i = LBound(zahl) To UBound(zahl))
    zahl(i) = 5
  Next i
End Sub
```

- Dieser Prozedur wird ein Array `zahl` unbekannter Grösse übergeben.

- Die vollständige Syntax von LBound und UBound lautet

- `LBound(Arrayname [, Dimension]`
  `UBound(Arrayname [, Dimension]`

- Beispielsweise ermittelt der Ausdruck `UBound(x, 1)` die obere Grenze `10` der ersten Dimension des als Beispiels verwendeten `Arrays x`, `UBound(x, 2)` die obere Grenze `3` der zweiten Dimension und `UBound(x, 3)` die obere Grenze `9` der dritten Dimension.

ReDim und Preserve

- Die Grösse von mit `Dim` ohne Indexangaben deklarierten ”dynamischen” Arrays können Sie mit einer `ReDim`-Anweisung beliebig festlegen und mit nachfolgenden `ReDim`-Anweisungen nachträglich verändern.

```
Sub Test
  Dim x() As Single
  Anweisungen
  ReDim x(1 To 10) As Single     '(Neu dimensionieren)
  Anweisungen
  ReDim x(23 To 54) As Single    '(Neu dimensionieren)
  Anweisungen
End Sub
```

- Jede `ReDim`-Anweisung erzeugt das Array neu. Die zuvor in den einzelnen Variablen gespeicherten Werte gehen dabei verloren. Wollen Sie sie erhalten, benötigen Sie den Zusatz `Preserve`
- `ReDim Preserve x(5 To 20) As Single`
- Die Inhalte der Variablen `x(1)` bis `x(4)` gehen verloren. `x(5)` bis `x(10)` bleiben erhalten. `x(11)` bis `x(20)` haben zunächst den Inhalt 0.

Felder löschen

- `Erase` (Löschen) löscht den Inhalt der Elemente von statischen Feldern

Array-Funktion

- Die Funktion `Array` übergibt einen Funktionswert vom Typ `Variant`, der ein Array enthält:
- `Datenfeld = Array(Argumentenliste)`

- Beispielsweise übergibt der Ausdruck:
  `Dim x As Variant`
  `x = Array(22, 15, 35)`
  in `x` ein `Array` mit den drei Werten `22, 15` und `35`. Obwohl `x` eine einzelne `Variant`-

Variable ist und kein `Array`, können Sie auf die darin enthaltenen Elemente wie auf ein `Array` zugreifen und beispielsweise mit:

- `y = x(2)`
  der Variablen `y` das letzte Element des in `x` enthaltenen Arrays zuweisen.

Mehrdimensionale Arrays

- `Dim Arrayname(Dimension1, Dimension2, …) As Datentyp`

- Zweidimensionales Array

- `Dim x(1 To 10, 1 To 3)`
- Vergleichen Sie ein zweidimensionales Array mit einer Tabelle, die in Spalten und Zeilen unterteilt ist.
- `x(3,2) = 20 '(dritte Spalte, zweite Zeile = 20)`

- Dreidimensionale Arrays

- ReDim x(1 To 10, 1 To 3, 5 To 9)
- Dieses dreidimensionale `Array` mit 150 Elementen (10*3*5 Variablen) kann man sich mit räumlichen Vorstellungsvermögen noch als Würfel vorstellen.

- VBA ermöglicht Arrays mit **maximal 60 Dimensionen**. Die Gesamtzahl der Elemente (Variablen) hängt vom verfügbaren Speicherplatz ab.

Parameter Array

- Sie können einer Prozedur *beliebig viele nacheinander aufgeführte Argumente* als ein "**Parameter-Array**" übergeben. Dazu muss die Prozedur folgendermassen deklariert sein:

  - `Sub Testproc(Nachname As String, ` **`ParamArray`** `x() As Variant()`

  Der Aufruf dieser Prozedur erfolgt z.B. so:

  - `Call Testproc("Maier", 5, 8, 17, 29, 2)`

- Das Schlüsselwort **`ParamArray`** vor dem zweiten Argument bewirkt, dass VBA das zweite übergebene Argument und alle folgenden Argumente zu einem Array zusammenfasst und dieses Array der Arrayvariablen `x` übergibt.
- x muss die letzte Prozedurvariable sein, der keine weitere Argumentdeklarationen folgen darf! Ausserdem muss `x` vom Typ **`Variant`** sein!

For Each-Schleife für Arrays

- Die `For..Each`-Schleife ist eine Sonderform der For..Next-Schleife und zum konfortablen "Durchlaufen" von Arrays und den später erläuterten "Klassen" gedacht.

```
For Each Element in Group
  [Block]
Next Element

Public Function ForEach_loop()
  Dim zahlen(1 To 10) As Single, i As Integer, x As Variant
```

- Mit For

```
  For i = 1 To 10
```

```
    Zahlen(i) = Rnd(1)
  Next i
```

- Mit For Each

```
  For Each x In zahlen()
    Debug.Print x
  Next x
End Function
```

- Der Vorteil gegenüber For..Next: Sie müssen sich nicht darum kümmern, welchen Index das erste bzw. das letzte Element des Arrays besitzt.

DefBool, DefCur, DefDbl, DefDate, DefInt, DefLng, DefObj, DefSng, DefStr, DefVar

- Können Variablen mit bestimmten Anfangsbuchstaben anders voreinstellen. Die Kommandos müssen am Beginn eines Moduls angegeben werden:

```
DefSng a-f
DefLng g, h
```

- Alle Variablen, die mit `a` bis `f` bzw. mit `g` oder `h` anfangen, weisen jetzt den Defaultdatentyp `Single` bzw. `Long` auf. Der Defaultdatentyp gilt nur für jene Variablen, bei denen im `Dim`-Befehl nicht explizit ein anderer Datentyp angegeben wurde.

Longzahlen beim rechnen explizit als soche kennzeichnen

- `Dim l As Long`
- `l = 255&  * 256`
- Damit Excel erkennt, dass es die Multiplikationsroutine für `Long`-Zahlen verwenden soll. Sonst werden die Zahlen intern als Integerzahlen interpretiert, was intern zu einer überschreitung kommt und führt daher schon vor der Zuweisung an l zur Fehlermeldung.

Zuweisung an Feld besser mit Array

Anstatt:

```
a(0)=1: a(1)=7: a(2)=3 etc.
```

besser:

```
a = Array(1, 7, 3)
```

Dim x

```
x = Array(10, 11, 12)
Debug.Print x(1)      'liefert 11
```

22.1 Syntaxzusammenfassung

Umgang mit Variant-Variablen:

IsNumeric(variable)	IstZahl
IsDate(variable)	IstDatum / IstUhrzeit
IsObject(variable)	IstObjekt
IsError(variable)	IstFehlerwert
IsEmpty(variable)	IstLeer
IsNull(variable)	IstInizialisiert
VarType(variable)	numerischer Wert, der den Datentyp angibt

TypeName(variable)	Zeichenkette, die Daten-/Objekttyp beschreibt

Felder:

Option Base 1	kleinster zulässiger Index ist 1 (statt Default 0)
Dim feld1(5), feld2(10, 10)	ein- und zweidimensionales Feld
Dim feld3(-3 bis 3)	Feld mit negativen Indizes
Dim feld4()	vorläufig leeres Feld
Redim feld4(10)	dynamische Neudimensionierung
Redim Preserve Feldr(20)	wie oben, aber ohne Daten zu löschen
Erase Feld()	löscht das Feld
LBound(feld()	ermittelt den kleinsten erlaubten Index
UBound(feld()	ermittelt den grössten erlaubten Index
L/UBound(feld(), n)	wie oben, aber für die n-te Dimension

Datenfeld

Dim x	normale Variant-Variable
x = Array(x1, x2, …)	Zuweisung

23 Konfigurationsdateien, individuelle Konfiguration

Optionen

- Die Bedeutung der Optionen in Excel im Menü Extras / Optionen sind offensichtlich oder können der Onlinehilfe entnommen werden.
- Die Datei und Druckoptionen gelten nur für die aktive Datei. Es ist anders bei Word nicht ohne weiteres möglich, Excel so einzustellen, dass es beim Speichern immer eine Sicherungskopie verwendet, beim Drucken generell links einen 4 cm breiten Rand frei lässt etc. Eine mögliche Lösung dieses Problems stellen Mustervorlagen dar (Kapitel 9).
- Einige **Virenschutzoptionen** sind im Dialog Extras / Makros / Sicherheit versteckt. Wo diese Einstellung gespeichert werden, ist zum Glück nicht dokumentiert (auf jeden Fall nicht ian den hier beschreibenen Orten). Eine Veränderung dieser Optionen per VBA-Code ist nicht vorgesehen. (Es werden sich aber zweifellos findige Programmierer finden, die auch das mit dem Aufruf einiger API-Funktionen bewerkstelligen.)

Optionen per Programmcode

- Die Einstellung der meisten Excel-Optionen erfolgt über zahllose Eingenschaften des Objekts `Application`. Optionen, die nicht Excel als Ganzes betreffen, können aber die Eigenschaft des jeweiligen Objekts verändert werden, wobei die Zuordnung nicht in jedem Fall logisch ist.
- Der aktive Drucker wird über die `ActivePrinter`-Eigenschaft des `Application`-Objekts eingestellt. Es gibt allerdings keine Möglichkeit, per VBA-Code eine Liste der zur Verfügung stehenden Drucker zu ermitteln.

Es folgt ein Überblick über die wichtigsten Eigenschaften und Methoden:

Applications-Object (allgemeine Optionen)

ActivePrinter	Einstellung dtenes momentan gültigen Druckers
AddIns(…)	Zugriff auf Add-Ins
Calculation	Neuberechnung von Tabellen automatisch / manuell
CommandBars(…)	Zugriff auf Menü- und Symbolleis
DisplayAlerts	Warnungen anzeigen
DisplayFormulaBar	Bearbeitungsleiste ein / aus (*True*/*False*)
DisplayFullScreen	Modus ganzer Bildschirm ein / aus
DisplayNoteIndicators	rote Markierungspunkte in Zellen mit Notizen anzeigen
DisplayStatusBar	Statuszeile
MoveAfterReturn	Cursor springt mit Return in nächste Zelle einer Tabelle
MoveAfterReturnDirection	Richtung der Cursorbewegungf durch *Return*
OnEvent …	diverse Ereignisprozeduren
PromtForSummaryInformation	Dialog zur Eingabe von Informationen beim Speichern
ScreenUpdating	Bildschirmaktualisierung während Makroausführung
SheetsInNewWorkbook	Anzahl der leeren Tabellenblätter
StandardFont	Name des Defaultzeichensatzes in Tabellen
StandardFontsize	Grösse des Defaultzeichensatzes in Tabellen

Workbook-Objekt (dateispezifische Optionen)

ChangeFileAccess	Zugriff ändern
Colors	Zugriff auf die Farbpalette (56 Farben) der Datei
CreateBackup	beim Speichern Backup-Datei erzeugen

DisplayDrawingObject	Zeichnungsobjekte anzeigen
Protect	Schutz auf Formatvorlagen
Styles(…)	Datei sichtbar / unsichtbar (ausgeblendet)
Visible	

Worksheet-Objekt (tabellenblattspezifische Optionen)

DisplayAutomaticPageBreaks	Seitengrenzen in den Tabellenblättern anzeigen
EnableAutoFilter	lässt die Anzeige von Autofiltern zu
EnableOutlining	lässt die Anzeige von Gliederungen (Gruppierungen) zu
EnablePivotTable	lässt das Erstellen von Pivottabellen zu
FilterMode	Autofilter an / aus
PageSetup	Zugriff auf Seiten- und Druckereinstellungenj
SetBackgroundPicture	Hintergrundbild einstellen
Visible	Arbeitsblatt sichtbar / unsichtbar

Window-Object (fensterspezifische Optionen)

DisplayFormulas	Formeln statt Ergebnisse anzeigen
DisplayGridlines	Gitterlinien anzeigen
DisplayHeadlings	Zeilen- und Spaltenköpfe anzeigen
DisplayHorizontalScrollBar	horizontale Bildlaufleiste anzeigen
DisplayOutline	Gliederung (Gruppierung) anzeigen
DisplayZeros	0-Werte anzeigen (oder leere Zelle anzeigen)
DisplayVerticalScrollBar	vertikale Bildlaufleiste anzeigen
DisplayWorkbookTabs	Blattregister anzeigen
FreezePanes	geteiltes Fenster fixieren / nicht fixieren
DisplayPanes	Farbe der Gitternetzlinien einstellen (RGB-Wert)
GridLineColor	Farbe der Gitternetzlinein aus Farbpalete (0 bis 55)
GridLineColorIndex	Zugriff auf Seiten- und Druckereinstellungen
PageSetup	Fenster geteilt / nicht geteilt
Split	Spalte, in der das Fenster geteilt ist
SplitColumn	Zeile, in der das Fenster geteilt ist
TabRation	Verhältnis Blattregister / horizontale Bildlaufleiste
Zoom	Zoomfaktor

PageSetup-Object (Seitenlayout, wird für jades Blatt gesondert eingestellt)

BlackAndWhite	Ausdruck in Schwarzweiss
BottomMargin	unterer Rand in Punkt (0.35)
CenterFooter	Fusszeile, mittlerer Teil
CenterHeader	Kopfzeile, mittlerer Teil
CenterHorizontal	Ausdruck horizontal zentrieren
CenterVertical	Ausdruck vertikal zentrieren
FirstPageNumber	Anfangszahl für Seitennumerierung
FooterMargin	Platz für Fusszeile
HeaderMargin	Platz für Kopfzeile
LeftFooter	Fusszeile, linker Teil
LeftHeader	Kopfzeile, linker Teil
LeftMargin	linker Rand in Punkt (0.35)
Orientation	Druck im Hoch- oder Querformat
PaperSize	Papiergrösse
PrintArea	zu druckender Tabellenbereich

PrintTitleColumns	Spaltenbeschriftung (wird auf jedem Blatt gedruckt)
PrintTitleRows	Zeilenbeschriftung (wird auf jedem Blatt gedruckt)
RightFooter	Fusszeile, rechter Teil
RightHeader	Kopfzeile, rechter Teil
RightMargin	rechter Rand in Punkt (0.35)
TopMargin	oberer Rand in Punkt (0.35)

DefaultWebOption (Excel global) / WebOptions (dateispezifisch)

AllowPNG	Bilder im PNG-Format codiert
DownloadComponents	evt. Fehlende Web-Komponente übertragen
Encodin	gewünschter Zeichensatz
LocationOfComponents	Ort, an dem Web-Komponenten gespeichert sind
OrganizeInFolder	Bilder etc. in eigenem Verzeichnis speichern
RelyOnCSS	*Cascading Style Sheets* verwenden
RelyOnVML	*Vector Markup Language* verwenden

Konfigurationsdateien

- Die benutzerspezifischen Konfigurationsdateien von Excel werden in Unterverzeichnissen des persönlichen Verzeichnisses gespeichert. Wir kürzen es mit **Userprofile** ab.
- Dateinamen und Pfad von Konfigurationsdateien ändern sich mit jeder Version – nicht zuletzt, um Konflikte durch die gelichzeitige Verwendung mehrerer Office-Versionen zu vermeiden. Wenn Sie global Excel-Anwendungen programmieren möchten, dürfen Sie sich also nicht darauf verlassen, dass sich die Konfigurationsdateien an einem bestimmten Ort befinden.
- Excel verstreut Informationen über die aktuelle Konfiguration und die Einstellung von Optionen über die ganze Festplatte. Die Fülle der Konfigurationsdateien wird von Version zu Version unübersichtlicher:

- Eingie individuelle Einstellungen werden in der Windows-Registrierdatenbank gespeichert.

- Informationen über den Inhalt und die Plazierung der Symbolleiste befindet sich in ...**Excel.xlb**
- Die persönliche Makroarbeitsmappe wird in ...**xlstart\Personl.xls**
- Global verfügbare Makros können in beliebigen Dateien im Verzeichnis ...**xlstart** gespeichert werden
- Persönliche Mustervorlagen werdein in ...**Vorlagen** gespeichert
- Für globale Mustervorlagen ist das Verzeichnis **...\Xlstart** vorgesehen. (Das Verzeichnis Office2000\Templates\1031 das offensichtlich das Vorlagenverzeichnis ist, hat für Excel 2000 keine Bedeutung – zumindest nicht in der aktuellen Version. 1031 ist der Sprachcode für die deutsche Version.
- Globale Add-In-Dateien werdein in ...**Makro** gesepeichert
- Persönliche Add-In-Dateien befinden sich dagegein in ...**AddIns**
- Vordefinierte (also mit Excel mitgelieferte) Diagrammvorlagen werdein in ...**XL8galry.xls** gespeichert
- Selbstdefinierte Diagrammvorlagen werden in ...**Xlusrgal.xls** gespeichert
- Alle verbleibenden Einstellungen sind dateispezifisch und werden in der eigentlichen Excel-Date gespeichert.

Informationen zur Symbolleiste

Die Datei ...**Excel.xlb** wird für jeden Anwender automatisch angelegt, sobald zum ersten Mal eine Veränderung an der Symbolleiste durchgeführt wird. Die Datei enthält Informationen zur Anordnung der Symbolleisten und des Tipassistenten, zu Veränderungen an den vor-

handenen Symbolleisten, Pfade zu den zhugeordneten Makrofunktionen sowie neue Symbolleisten, die beim letzten Verlassen von Excel verfügabar waren.

*.xlb-Dateien können durch Datei / Laden geladen werden und verändern dann den aktuellen Zustand der Symbolleisten. Der zuletzt gültige Zustand wirdd beim Verlassen von Excel automatisch gespeichert. Es besteht aber keine Möglichkeit, die Datei durch ein Menü- oder Makrokommando zu speichern, ohne Excel gelichzeitign zu verlassen.

Es besteht die Möglichkeit, selbstdefinierte Symbolleisten mit Ansicht / Symbolleisten / Anpassen / Symbolleisten / **Anfügen** unmittelbar in einer Excel-Datei zu speichern. Das ist dann sinnvoll, wennn die Symbolleiste auch anderen Anwendern (womöglich auf einem anderen Rechner) zur Verfügung stehen soll.

24 Mustervorlagen

24.1 Mustervorlagen mit Datenbankanbindung

- Der Vorlagenassistent wird mit Extras / Add-In-Manager installiert. Wenn der Add-In-Manager nichts vom **Vorlagenassistent** weis, müssen Sie den Assistenten nachinstallieren). Der Vorlagenassistent stellt eine Verbindung zwischen einer Mustervorlage und einer Datenbankdatei her.

Bedienung des Vorlagenassistenten

- Das Format kann eine Excel-Tabelle oder eine Access-Datei sein.
- Nach der Definition der Datenbankdatei markieren und beschriften Sie jene Eingabe- oder Ergebniszellen, die in der Datenbank gespeichert werden sollen.
- Sie haben die Möglichkeit, bereits vorhandene Excel-Dateien, die dem Format der Mustervorlage entsprechen, in die Datenbank aufzunehmen.
- Schliesslich können Sie noch eine Verteilerliste E-Mail-Adressen angeben. Jedesmal, wenn eine *neue* Datei auf der Basis der Mustervorlage geschlossen wird, erscheint eine Frage ob diese Datei an die Verteileradresse versandt werden soll.

Verwendung der Datenbankmustervorlage

- Der Anwender lädt die Excel-Muserevorlage, füllt die vorgesehenen Felder aus uns speichert die Datei. Beim Speichern erscheint automatisch ein Dialog, in dem der Anwender gefragt wird, ob die Daten in einer Datenbankdatei gespeichert werden sollen. Wenn der Anwender diese Frage bejaht, werden alle relevante (angegebenen) Zellen in eine neue Zeile bzw. einen neuen Datensatz der Datenbank eingetragen.
- Im Prinzip wird die Mustervorlage also wie bisher verwendet: ausfüllen, speichern, drucken. Neu ist nur, dass die Daten einer ausgewählter Zelle *zusätzlich* in einer speziellen Datei gespeichert werden.
- **Besonders attraktiv** sind Mustervorlagen mit Datenbankanbindung in Netzwerken. In dem allen Anwendern dieselbe Vorlage zur Verfügung gestellt wird, kann erreicht werden, dass alle wesentlichen Daten von Formularen, die auf dieser Vorlage erstellt worden sind, automatisch in einer zentralen Datei eingetragen werden.

Interna

- Die Mustervorlage wird durch zwei ausgeblendete Blätter ergänzt:

- Ein Excel-4-Makro-Blatt enthält ein Makro, das beim Laden der Datei automatisch ausgeführt wird. Dieses Makro lädt eine Add-In-Datei. Das Add-In enthält den eigentlichen VBA-Programmcode für die Verwaltung der Datenbankdatei. (Die Trennung zwischen Mustervorlage und Code in zwei eigene Dateien hat den Vorteil, dass die aus Mustervorlagen resultierenden Excel-Dateien nicht unnötig aufgebläht werden.)
- Sie können sich den Excel-4-Makrocode ansehen, wenn sie das Blatt vorher in der Entwicklungsumgebung sichtbar machen:

```
Sheets(”AutoOpen Stub Dada”).Visible = True
```

- Das andere Tabellenblatt kann ebenfalls in der Entwicklungsumgebung sichtbar gemacht warden:

Sheets(”TemplateInformation”).Visible = True

- Es enthält Informationen darüber, welche Zellen von welchen Blatt der Mustervorlage wo gespeichert werden sollen.

Datenbankanbindung löschen

- Es besteht keine Möglichkeit, diese Verbindung wieder zu entfernen. Sie müssen die betreffenden Tabellenblätter in eine neue Excel-Datei kopieren und dann diese als neue Vorlage speichern.

24.2 Probelme bei einer Mustervorlage

Rechnungsnummer

- Die Rechnungsnummer müsste an einer zentralen Stelle in einer eigenen Datei gespeichert werden. Von dort wird die Nummer erst unmittelbar vor dem Drucken gelesen und sofot um eins vergrössert. Die Gefahr eines gleichzeitigen Zugriffs zweier Excel-Anwendungen auf die gemeinsame Rechnungsnummerndatei ist damit zwar noch immer gegeben, aber schon recht unwahrscheinlich. Ganz professionell lässt sich das Problem nur lösen, wenn die Verwaltung der Rechnungsnummern durch ein zentrales Programm erfolgt (am besten durch eine Datenbankanwendung, die auch andere Rechnungsdaten speichert).

Daten wieder einspeisen

- Es müssten Prozeduren erstellt werden, mit der die Daten einer bereits gespeicherten Rechnung wieder eingelesen, verändert und ausgedruckt werden können. Hier tritt natürlich auch ein Sicherheitsproblem auf: Soll es generell möglich sein, zwei Rechnungen mit derselben Rechnungsnummer auszudrucken? Oder muss die fehlerhafte Rechnung storniert und die korrigierte Rechnung als neue Rechnung ausgedruckt werden?

25 Datenverwaltung in Excel

25.1 Grundlagen

Was ist eine Datenbank

- In Datenbanksysteme können grössere Datenmengen viel sicherer und effizienter als in Excel verwaltet werden.
- Client / Server-System: Hier erfolgt eine Trennung zwischen dem Programm, das die Daten verwaltet (der Server), und dem Programmen, die auf die Daten zugreifen (Clients). Beliebte Datenbankserver sind Oracle, Mircrosoft SQL Server, IBM DB/2, Informix uns Sybase. Zur Programmierun der Clients kann z.B. Visual Basic oder Delphi eingesetzt werden.

- Sie können das Zusatzprogramm MS-Query zum Einlesen externer Daten oder die ADO-Bibliothek zur Datenbankprogrammierung verwenden.

25.2 Excel versus Datenbanksystem

Unterschiede zwischen Tabellenkalkulationsprogrammen und richtigen Datenbanksystemen

- Excel lädt alle Daten (also die gesamte Datei) in den Arbeitsspeicher und steht dort unmittelbar und verzögerungsfrei zur Verfügung.
- In Datenbanksystemen bleiben die Daten generell in der Datenbankdatei. In den RAM werden immer nur möglichst kleine Portionen dieser Daten geladen. Jede Veränderung der Daten wird sofort gespeichert

- In Excel kann eine Datei nur als ganzes gespeichert werden, und das dauert bei grösseren Datenmengen so lange, dass dieser Vorgang viel zu selten durchgeführt wird. Im Gegensatz dazu wird in einem Datenbanksystem jede noch so kleine Änderung sofort gespeichert.

Merkmale von richtigen Datenbanken, die in Excel fehlen

- Excel kennt keine Indizes
- Excel kennt keine Relationen
- Excel fehlen Schutzmechanismen gegen das Löschen oder Verändern von Daten
- Excel fehlt ein Berichtsgenerator. Versuchen Sie z.B. mal mit Excel Etiketten zu bedrucken
- Excel ist nicht zur Verwaltung vernetzter Daten konzipiert.

Warum wird Excel dennoch oft als Datenbank verwendet?

- Excel ist einfach zu bedienen
- Excel steht schon zur Verfügung
- Excel stellt einfache Datenbankfunktionen zur Verfügung und ist daher für kleine Datenbankanwendungen durchaus geeignet
- Excel stellt auch bei der Verwendung als Datenbanksystem alle Tabellenfunktionen zur Verfügung. Richtige Datenbanksysteme können in dieser Hinsicht nicht mithalten.
- Excel stellt als Tabellenkalkulationsprogramm zur Zeit den Standard im Bürosektor dar. Einen vergleichbaren Standard im Datenbanksektor gibt es nicht. Aus diesem Grund eignen sich Excel-Dateien ausgezeichnet zum Datenaustausch zwischen Mitarbeitern eines Betriebs bzw. verschiedenen Firmen.

Die Konsequenz: Daten extern speichern, Datenanalyse in Excel

- Die optimale Lösung besteht oft darin, dass Sie das beste aus beiden Programmen kombinieren
- - Das Zusatzprogramm MS-Query ermöglicht den interaktiven Zugriff auf fas jedes Datenbanksystem.
- - Die Objektbibliothek ADO ermöglicht den Datenbankzugriff auch per Programmcode. Über die Möglichkeiten von MS-Query hinaus **können mit der ADO-Bibliothek auch Daten verändert oder neue Datensätze gespeichert werden**.

Wann sind Datenbanken direkt in Excel dennoch sinnvoll?

Wenn:

- - die Datenmenge klein ist (Tabellen mit bis zu 1000 Zeilen sind einigermassen unprobelmatisch)
- - Die Daten sehr einfach strukturiert sind (keine Notwendigkeit von Relationen, geringe Redunanz) und

- - keine Notwendigkeit einer gemeinsamen Bearbeitung der Daten via Netzwerk besteht.
- Grobfahrlässig wäre es, wirklich lebenswichtige Daten – etwas im medizinischen Bereich – mit Excel zu verwalten!

25.3 Datenverwaltung in Excel

- Wenn in einzelnen Spalten fallweise sehr umfangreiche Informationen stehen, dann sollten Sie für die gesamte Spalte das Attribut ”Zeilenumbruch” aktivieren. Gleichzeitig sollten Sie die gesamte Tabelle vertikal nach oben ausrichten
- Bei der Formatierung von Zellen ist es generell sinnvoll, nicht einzelne Zellen, sondern immer ganze Spalten oder überhaupt die gesamte Tabelle zu formatieren.

- Die Berechnungsformel JAHR(0) ist zwar einfach, aber nicht ganz exakt. Durch Schaltjahre kann es einen Tag vor oder nach dem Geburtstag vorkommen, dass ein Jahr falsches Alter angezeigt wird.

Datenmaske

- Datenbankspalten, in denen gerechnete Formeln (und nicht eingegebene Werte) stehen, können in der Maske nicht verändert werden.
- Die Funktion Bearbeiten / Rückgängig steht nach dem Ende des Datenbankdialogs nicht zur Verfügung.

Vorteile

- In der Datenmaske werden alle Daten eines Datensatzes kompakt angezeigt. Datenbanktabellen sind häufig so breit, dass Sie nur einen Ausschnitt sehen können
- Die Bedienung setzt keine Excel-Kenntnisse voraus
- Eine unbeabsichtigte Zerstörung von Daten ist weitgehend ausgeschlossen

Nachteile

- Filterkriterien werden nicht berücksichtigt
- Zahlreiche Datenbankkommandos – etwa das Neusortieren – können nur ausgeführt werden, indem die Maske verlassen wird.
- Der Aufbau der Maske ist starr vordefiniert. Es ist nicht möglich, die Maske so einzurichten, dass nur ausgewählte Datenfelder verändert werden können.
- Eine automatische Kontrolle der Eingabe (etwa ob im Geburtsdatumsfeld tatsächlich ein gültiges Datum eingegeben wurde) ist nicht möglich.
- Gleichzeitiges Bearbeiten mehrerer Datensätze (etwa zum Löschen aller veralteten Datensätze) ist nicht möglich.
- Beim Verlassen der Datenmaske wird kein Wert an den VBA-Code zurückgegeben, der Aufschluss über den zuletzt dargestellten Datensatz gibt.

Datenmaske anzeigen

- `ShowDataForm` zum Aufruf der in Excel vordefinierten Datenbankmasken funktioniert leider nicht ganz optimal: Das Kommando nimmt an, dass die Datenbank mit der Zelle A1 beginnt – unabhängig davon, wo der Zellzeiger gerade steht. Dem können Sie abhelfen, wenn Sie dem Zellbereich mit der Datenbank den Namen ”database” (im deutschen Excel alternativ ”Datenbank”) zuweisen und anschliessend `ShowDataForm` ausführen. Das Beispiel unten geht davon aus, dass A5 eine Zelle der Datenbank ist.

```
ActiveSheet.Range(”A5”).CurrentRegion.Name = ”database”
ActiveSheet.ShowDataform
```

Doppelte Sätze entfernen und von Spalte A nach Spalte B bringen mit Spezialfilter

```
Sub DoppelteSätzeErmitteln()
Dim Blatt As Worksheet
Dim Zelle As String
 Set Blatt = ActiveSheet
 On Error Resume Next
 Blatt.Activate
 Zelle = Range("A2", Range("A2").End(xlDown)).Address
 Range(Zelle).AdvancedFilter Action:=xlFilterCopy, _
 CopyToRange:=Range("B2"), Unique:=True
End Sub
```

Autofilter und Serienbrief

- Da Word nur die erste Tabelle für den Serienbrief erkennt, empfiehlt es sich, die Datenbank auf der zweiten Tabelle zu haben und die mit Autofilter gefilterten Daten (Sichtbare Zellen) per Makro auf die erste Tabelle zu kopieren.

25.4 Syntaxzusammenfassung

Datenbankverwaltung in Excel (Sortieren, Gruppiern etc.)

rng.Name = "database"	benennt Bereich für *ShowDataForm*
wsh.ShowDataForm	zeigt die Datenbankmaske an
rng.Sort …	sortiert die Datenbank
rng.Find …	nach Daten sortieren
rng.FindNext	nochmals suchen
rng.FindPrevious …	rückwärts suchen
rng.Replace …	suchen und ersetzen
rng.Consolidate …	mehrere Tabellen konsolidieren

Filter

rng.Autofilter …	Autofilter aktivieren
rng.AdvancedFilter	Spezialfilter aktivieren
wsh.FilterMode	gibt an, ob die Tabelle gefilterte Daten enthält oder nicht
wsh.AutoFilter	verweist auf das *AutoFilter*-Objekt
wsh.AutoFilter.Filters(…)	verweist auf dessen *Filter*-Objekt (mit Filterkriterien)
wsh.AutoFilterMode	gibt an, ob Autofilter aktiv ist
wsh.AutoFilterMode = False	deaktiviert Autofilter
wsh.ShowAllData	entfernt Filterkriterien

26 Tipps und Tricks

26.1 Laufzeiten verkürzen

Berechnungen und Bildschirmaktualisierung

```
Application.Screenupdating = False
Application.Calculation = xlCalculationManual
```

`Calculation` muss am Ende der Prozedur wieder einschaltet werden

```
Application.Calculation = xlCalculationAutomatic
```

- Sollten Sie während des Makros auf sich ändernde Zellen zugreifen wollen, dann müssen Sie schon während der Laufzeit des Makros diese Zellen gezielt aktualisieren. Dazu setzen Sie den Befehl

```
Range(”A1”).Calculate
```

- Ein. Damit wird nur die Zelle A1 aktualisiert. Diese Aktualisierung können Sie natürlich auch auf ganze Beriche ausdehnen

```
Range(”Bereichsname”).Calculate
```

- Bzw. auch für Spalten und Zeilen:

```
ActiveSheet.Rows(2).Calculate
ActiveSheet.Columns(2).Calculate
```

- Auch das Aktualisieren einzelner Tabellenblätter ist über den Befehl

```
Worksheet(1).Calculate
```

- jederzeit möglich. Wenn Sie lediglich den Befehl

```
Application.Calculate
```

- Angeben, werden alle geöffneten Arbeitsmappen neu berechnet

Befehle zusammenfassen

- Sie müssen versuchen, mit so wenig Befehlen wie möglich auszukommen.

Prozedur: Zellen übertragen langsam

```
Sub ZellenÜbertragenLangsam()
Dim i As Integer
Dim s As String
  Sheets(”Tabelle1”).Activate
  Range(”A1”).Select
  s = Range(”A1”).Value
  For i = 1 To 10
    Sheets(”Tabelle2”).Activate
    ActiveCell.Value = s
    ActiveCell.Offset(1, 0).Select
    s = ActiveCell.Value
```

```
  Next i
End Sub
```

Was kann man verbessern?

1. Im ersten Schritt sollten die Namen der beteiligten Tabellenblätter gleich zu Beginn des Makros festgelegt werden. Dies erleichtert später eine Änderung des Quellcodes. Wenn sich beispielsweise die Namen der Tabellenblätter ändern, müssen Sie nicht im gesamten Code suchen, um die Namen der entsprechenden Tabellenblätter anzupassen.
2. 2. Im zweiten Schritt machen sie das Makro flexibler. Das Makro hört nach genau zehn Zeilen auf, Daten zu übertragen. Was aber passiert, wenn mehr als zehn Zellen übertragen werden sollen?
3. 3. Im dritten Schritt versuchen sie, weniger zwischen den Tabellenblättern hin- und her zu springen. Auch das Verschieben des Zellenzeigers sollte nicht so häufig geschehen. Wenn Sie öfters auf andere Tabellenblätter wechseln müssen, sollten Sie auf jeden Fall die Bildschirmaktualisierung mit dem Befehl *Application.Screenupdatin = False* ausschalten, um nicht ein Schwindelgefühl beim Anwender zu erzeugen bzw. überhaupt gar nicht zwischen den einzelnen Tabellen hin- und herspringen.

Prozedur: Zellen übertragen schnell

```
Sub ZellenÜbertragen()
Dim Tab1 As Worksheet
Dim Tab2 As Worksheet
Dim i As Integer
Dim y As Integer
  Set Tab1 = Worksheets(”Tabelle1”)
  Set Tab2 = Worksheets(”Tabelle2”)
  For i = 1 To Tab1.UsedRange.Rows.Count
    y = y + 1
    Tab2.Cells(i, 1) = Tab1.Cells(y, 1)
  Next i
End Sub
```

- Bei dieser Methode verwenden Sie die Eigenschaft *Cells*, bei der Sie jeweils einen Zeilenindex und einen Spaltenindex angeben müssen. Da sich die zu übertragenden Daten in Spalte A befinden, könnenm Sie den Spaltenindex in der Schleife konstant auf dem Wert 1 halten. Den Zeilenindex y müssen Sie bei jedem Schleifendurchgang um 1 erhöhen.
- Wenn Sie eine Tabelle über den Namen ansprechen, benötigt Excel länger, als wenn Sie Tabellen über einen Index ansprechen.

Variablen und Konstanten einsetzen

- Vergessen Sie nicht, alle Variablen, die Sie im Makro verwenden, vorher zu definieren.
- Um noch mehr Schnelligkeit beim Interpretieren der einzelnen Befehle zu bekommen, können Sie sich auch überlegen, möglichst kurze Namen bei der Benennung von Variablen und Konstanten zu verwenden. Allerdings wird dadurch die Lesbarkeit des Codes ziemlich eingeschränkt.

Integrierte Tabellenfunktionen anwenden

- Die Anwendung der Tabellenfunktion Sum ist wesentlich schneller als das einzelne Addieren von Werten. So ist die folgende Zeile

```
Erg = Application.WorksheetFunction.Sum(Range("A1:H1"))
```

schneller als die nächsten drei Zeilen

```
For Each Zelle In Range("A1:H1)
  Erg = Erg + Zelle.Value
Next Zelle
```

- Sie können alle Tabellenfunktionen im Objektmanager einsehen (F2)
- Unter der Klasse "WorksheetFunction" finden Sie alle in Excel zur Verfügung stehenden Tabellenfunktionen
- Weitere Faktoren zur Geschwindigkeitssteigerung

Warnungen deaktivieren

```
Application.DisplayAlerts = False
```

Eingabe blockieren

```
Application.Interactive = False
```

- So ist Excel gegenüber allen Eingaben (Tastatur und Maus) blockiert. Im Regelfall ist das nicht erforderlich, weil Excel während der Makroausführung ohnedies keine Eingabe entgegennimmt.
- Die Eigenschaft `Interactive` muss am Ende der Prozedur unbedingt wieder auf `True` gesetzt werden. Es gibt keine Möglichkeit, die Eigenschaft ausserhalb des VBA-Codes zurückzusetzen.

Zeitaufwendige Berechnungen

- Während länger dauernder Berechnungen sollten Sie den Anwender durch einen Text in der Statuszeile über den Stand der Berechnung informieren. Das zeigt dem Anwender, dass der Rechner noch nicht abgestürzt ist.
- Die Statuszeile sollte zumindest Auskunft darüber geben, was der Rechner gerade tut. Noch besser ist es, wenn Sie ausserdem in einem Prozedurwert angeben können, wie weit die Berechnung bereits fortgeschritten ist.
- Text wird mit `Application.StatusBar` eingestellt.
- Eigenschaft `DisplayStatusBar` bestimmt, ob die Statusleiste angezeigt wird oder nicht. Wenn das nicht der Fall ist, können Sie die Statusleiste vorübergehend anzeigen und am Ende der Prozedur wieder verschwinden lassen.
- Leider besteht in Excel keine Möglichkeit, den Zustand einer länger andauernden Berechnung durch einen Statusbalken (mit den kleinen blauen Quadraten) anzugeigen. Excel verwendet dieses Gestaltungsobjekt zwar selbst häufig – etwa beim Laden und Speichern von Dateien – es fehlen aber VBA-Methoden zur Steuerung des Statusbalkens.

Statusleiste mit Prozentangaben

```
Sub StatusleisteMitProzentangabe()
  Const loopEnd = 1000000
  Dim statusMode As String
  nextUpdateTime As Date
  Dim i As String, x As String, result As String

  Application.EnableCancelKey = xlErrorHandler
```

```
  On Error GoTo Fehler
  nextUpdateTime = Now
  statusMode = Application.DisplayStatusBar
  'speichern, ob Statusleiste sichtbar
  Application.DisplayStatusBar = True
  'Statusleiste anzeigen

  For i = 1 To loopEnd
    If i Mod 50 = 0 Then
    'nur bei jedem 50. Durchlauf Zeit testen
      If Now > nextUpdateTime Then
      'Statuszeile regelmäßig aktualisieren
        nextUpdateTime = Now + TimeSerial(0, 0, 1)
        Application.StatusBar = _
         "Berechnung zu " & CInt(i / loopEnd * 100) & _
         " Prozent ausgeführt"
        DoEvents
      End If
    End If

  x = Sin(i) * Cos(i) ^ 3 * Sqr(i)          'damit wird Rechenzeit verbraucht
  x = Sin(i) * Cos(i) ^ 3 * Sqr(i)
  x = Sin(i) * Cos(i) ^ 3 * Sqr(i)
  x = Sin(i) * Cos(i) ^ 3 * Sqr(i)

  Next i

  Application.StatusBar = False
  'Steuerung an Excel zurückgeben
  Application.DisplayStatusBar = statusMode
  'alten Zustand wiederherstellen
  Beep
  Exit Sub

Fehler:
  If Err = 18 Then
    result = MsgBox("Soll das Programm fortgesetzt werden?",
vbYesNo)
    If result = vbYes Then Resume Next
  End If
  ' sonst Prozedur abbrechen
  Application.StatusBar = False
  'Steuerung an Excel zurückgeben
  Application.DisplayStatusBar = statusMode
  'alten Zustand wiederherstellen
  If Err = 18 Then Exit Sub
  Error Err
  'Fehlermeldung
End Sub
```

- Die Eigenschaft **EnableCancelKey** steuert das Verhalten von Excel beim Drücken von Strg+Untbr. Wenn `EnableCancelKey` auf `xlErrorHandler` gesetzt wird, tritt als Reaktion auf Strg+Untbr ein Fehler mit der Nummer 18 auf, der in einer Fehlerbehandlungsroutine aufgefangen werden kann.

26.2 Effizienter Umgang mit Tabellen

Effizientes Bearbeiten von Zellbereichen

- Wenn Sie eine grosse Anzahl von Zellen per VBA-Code bearbeiten müssen, besteht die langsamste (aber leider einfachste) Methode darin, jede Zelle einzeln anzusprechen. Daran ändert auch `ScreenUpdatin = False` und `Calculation = xlManual` nicht mehr viel. Wenn Sie schneller vorgehen möchten, haben Sie vier Möglichkeiten:

- Sie verwenden vordefinierte Excel-Methodem, wie `AutoFill` (automatisches Ausfüllen), `PasteSpecial` (Inhalte einfügen und dabei Operationen wie Subtraktion, Multiplikation etc. ausführen), `Copy` (Zellbereiche kopieren) etc. Solche Methoden sind *sehr* viel schneller im Vergleich zur herkömmlichen Programmierung

- Sie arbeiten mit Felder: Der Zugriff auf Feldelemente erfolgt viel schneller als der Zugriff auf Zellen. Felder können fertig ausgerechnet und dann als Ganzes in einen Zellbereich kopiert werden.

- Sie arbeiten mit Datenfeldern: Datenfelder haben gegenüber normalen Feldern zwar viele Nachteile, aber auch einen entscheidenen Vorteil: Sie können ganze Zellbereiche auf einmal in ein Datenfeld übertragen. (Bei normalen Feldern ist ein Datentransport nur in die umgekehrte Richtung möglich.)

- Sie arbeiten mit der Zwischenablage:

Arbeiten mit normalen Feldern

- Es ist kaum bekannt, dass die Inahalte von ein- und zweidimensionalen Feldern einfach per Zuweisung in einen Zellbereich kopiert werden können. Am einfachsten wird das anhand eines Beispiels verständlich:

```
Dim y(3)                                ’4 Elemente
y(i) = …
Worksheet(1).Range(”a1:d1”) = y         ‘verändert A1:D1

Dim x(9, 4)                             ‘10*5 Elemente
x(i, j) = …
Worksheets(1).Range(”a1:e10”) = x       ‘verändert die Zellen A1:E10
```

- Beim Umgang mit Feldern müssen einige Details beachtet werden:
- - Der Zielbereich muss exakt angegeben werden; wenn er kleiner ist als das Feld, werden entsprechend weniger Elemente übertrtragen; wenn er dagegen zu gross ist, werden die überzählige
- Zellen mit dem Fehlerwert `#NV` gefüllt.
- -Eindimensionale Felder können nur einem horizontalen Zellblock zugewiesen werden, nicht einem vertikalen.
- - Bei zweidimensionalalen Feldern gibt der erste Index die Zeile, der zweite Index die Spalte an, wie bei `Offset`.
- - Die Datenübertragung ist nur in der Richtung Feld – Tabelle möglich. Das Einlesen von Zellen in ein Feld ist nicht möglich (bzw. nur mit Datenfeldern – siehe etwas weiter unten)

- ’schnelle Variante, keiner als 1 Sekunde (10'000 Zellen ausfüllen)

```
Sub FastFill()
Option Base 1
Dim i As Double, j As Double, k As Double
Dim r As Range, r1 As Range, r2 As Range
Dim cells(200, 200)
  Worksheet(1).Activate
  Worksheet(1).Range("A1").CurrentRegion.Clear
  Application.ScreenUpdating = False
  Application.Calculate = xlManual
  For i = 1 To 200
    For j = 1 To 200
      k = i * 200 + j
      cells(i, j) = k
    Next
  Next

  'Zielbereich ermitteln
  Set r1 = Worksheet(1).Range("A1")
  Set r2 = r1.Offset(200, 200)
  r = cells
  Application.ScreenUpdating = True
  Application.Calculate = xlAutomatic
End Sub
```

Arbeiten mit Datenfeldern

- Das meiste, was oben für normale Felder besprochen wurde, gilt auch für Datenfelder. Neu ist, dass jetzt auch ein Datentransport aus einem Zellbereich in ein Datenfeld möglich ist.

```
Dim x As Variant
x = Worksheet(1).Range("A1:B4")       '8 Elemente lessen
…                                     'Bearbeiten
Worksheets(1).Range("C1:D4") = x      '8 Zellen verändern
```

- Auf die einzelnen Elemente kann nur in der Form `x(1, 1)` bis `x(4, 2)` (für B4) zugegriffen werden. Gegenüber normalen Feldern besteht folgende Unterschiede:
- - Der Zugriff auf das erste Feld beginnt mit dem Index 1. (Bei Feldern ist es normalerweise 0. Nur wenn sie `Option Base 1` verwenden, gilt auch bei Feldern der Index `1` als kleinster erlaubter Wert.
- - Die Grösse von Datenfeldern kann nicht im voraus durch Dim eingestellt werden. Die Anzahl der Elemente ergibt sich erst beim Kopieren von Zellen aus dem Tabellenblatt. Daher eignen sich Datenfelder vor allem dann, wenn bereits vorhandene Zellen verändert oder analysiert werden müssen. Echte Felder sind dagegen praktisch, wenn nur Daten in das Tabellenblatt geschrieben werden sollen.

Schutzmechanismen

- Einen Schutz, den Sie wieder auflösen können, kann auch von einem anderen Excel-Profi geknackt werden. Einigermassen sicher sind nur Schutzfunktionen, die durch Kennwörter abgesichert sind (und selbst die lassen sich knacken). Wirklich perfekt sind in dieser Hinsicht nur COM-Add-Ins, die als Binär-DLL weitergegeben werden – aber die weisen viele andere Nachteile auf (siehe Kapitel 14).
- Wirklich sicher ist Ihr Code nur in COM-Add-Ins

26.3 Hilfe zur Selbsthilfe

- Alles hat in diesem Buch nicht Platz. Es wird Ihnen nicht erspart bleiben, manchmal selbst zu experimentieren!

Nutzen Sie die Online-Dokumentation

- Insgesamt ist die Suche nach Detailinformationen um so leichter, je besser Sie Excel kennen bzw. VBA bereits kennen.

Experimentieren Sie im Direktbereich

- Sie können innerhalb des Direktfensters beinahe alle Sprachstrukturen von VBA verwenden – selbst Schleifen! Die einzige Bedingung besteht darin, dass die gesamte Anweisung in einer Zeile Platz haben sollte (bzw. mit "_" in mehreren).

26.4 Syntaxzusammenfassung

- Alle Eigenschaften und Methoden beziehen sich – wenn nichts anderes angegeben wird – auf das Objekt `Application`.

Hintergrundberechnungen, Optionen für die Programmausführung

Interactive = True/False	Eingaben zulassen oder nicht
EnableCancelKey = xlDisabled	keine Reaktion auf Strg+Untbr
.. = xlErrorHandler	Fehler 18 bei Strg+Untbr
DisplayAlerts = True/False	Warnmeldungen während der Makroausführung
DisplayStatusBar = True/False	Statusleiste anzeigen
StatusBar = "infotext"/False	Text in der Statuszeile einstellen

Geschwindigkeitsoptimierung

SceenUpdating = True/False	Bildschirmaktualisierung ein / aus
Calculation = xlAutomatic/xlManuel	automatische / manuelle Berechnung
Objekt.Calculate	Bereich / Blatt / ganze Anwendung neu berechnen

Rc steht fü reine Zeile oder eine Spalte (Row- oder Column-Objekt), *ws* für ein Tabellenblatt (Worksheet), *obj* für einen Zellbereich oder ein Zeichnungsobjekt (inklusive Steuerelemente und OLE-Objekte), *rng* für einen Zellbereich (*Range*-Objekt), *cb* für eine Symbolleiste (CommandBar-Objekt) und *wd* für eine Arbeitsmappe (Workbook-Objekt).

Schutzfunktion

rc.Hidden = Treu/False	Zeilen / Spalten aus- oder einblenden
ws.ScorllArea = "A1:E10"	Bewegungsradius einschränken
ws.Visible = True/False/xlVeryHidden	Blätter ein- oder ausblenden bzw. verstecken
obj.Locked = True/False	Objekt schützen (nur in Kombination mit *Protect*)
rng.FormulaHidden = True/False	Formeln verbergen
ws.Protect …	Blatt schützen (ausser Objekt mit *Locked = False)*
ws.Unprotect	Blattschutz aufheben
wb.Protect …	Aufbau der Arbeitsmappe schützen
wb.Unprotect	Schutz aufheben
cb.Protect = …	Symbolleiste schützen

Excel-4-Makros und Tabellenfunktionen

Run "makroname" [,para1, para2 …]	Excel-4-Makro ausführen
ExecuteExcelMacro "KOMMANDO(…)"	Excel-4-Makrokommando ausführen (deutsch)

WorksheetFunction.Function()	Tabellenfunktion ausführen (englisch)

Excel-Versionsnummer

Application.Version	Zeichenkette mit Excelversionennummer

27 Diverses

Den Office-Assistenten aufrufen

```
Sub OfficeAssistentAufrufen ()
Dim Karle As Object

Set Karle = Assistant.NewBallon
 With Karle
   .Heading = « Themen »
   .Icon = msoIconTip
   .Mode = msoModeAutoDown
   .BalloonType = msoBallonTypeButton
   .Labels(1).Text = « Strukturierte Programmierung »
   .Labels(2).Text = « Schnelle Programmierung »
   .Labels(3).Text = « Sichere Programmierung »
   .Animation = msoAnimationGreeting
   .Button = msoButtonSetNone
   .Show
 End With
End Sub
```

Textdatei als Objekt einfügen

```
Sub TextDateiAlsObjektEinfügen()
Dim Textobj As OLEObject
  On Error GoTo Fehler
  Set Textobj = ActiveSheet.OLEObjects.Add( _
    Filename:=”c:\eigene Dateien\Produkt.txt”, _
    Link:=True, _
    DisplayAsIcon:=False)
   Exit Sub
Fehler:
  MsgBox ”Die Textdatei konnte nicht gefunden werden!”
End Sub
```

Geben Sie dem Argument *Link* den Wert *True*, wenn Sie das eingefügte Objekt mit der originalen Datei verknüpfen möchten.

Add-In einbinden

Möchten Sie nun prüfen, ob ein bestimmtes Add-In schon verfügbar ist, bevor Sie mit VBA auf eine Funktion zugreifen, die in einem Add-In steckt, dann setzen Sie das folgende Makro ein.

```
Sub AddInPrüfenUndEinbinden()
With AddIns(”Automatisches Speichern”)
  If .Installed = False Then .Installed = True
     MsgBox ”Das Add In ” & .Name & ” ist nun verfügbar”
  End With
End Sub
```

Von Excel aus Word starten und ein Dokument öffnen

```
Dim wwobj As Object
  Set wwobject = CreateObject(”Word.Application”)
  wwobject.Visible = True
```

```
  wwobject.Documents.Open _
    Filename:=”F:\WordBuch\Notiz1.doc”, _
    ReadOnly:=True
  Set wwobject = Nothing
```

Mit der Anweisung `Set` wird das neue Objekt erstellt. Um eine OLE-Automation mit Word zu realisieren, verwenden Sie das Objekt `Word.Application`. Bei der Initialisierung der Variablen wird automatisch Word gestartet. Damit das Word-Fenster mit alen Symbolleisten und Menüs sichtbar wird, wird die Eigenschaft `Visibe` (Sichtbar) auf `True` (Wahr) gesetzt.

Beachten Sie ausserdem: Verwenden Sie (wie im Beispielfall) die Funktion `CreateObject`, wenn keine aktuelle Instanz des Objekts exisitert. Wird eine Instanz des Objekts bereits ausgeführt, so wird eine neue Instanz gestartet und ein Objekt des angegebenen Typs erstellt. Damit Sie die aktuelle Instanz verwenden oder die Anwendung starten und eine Datei laden können, verwenden Sie die `GetObject`-Funktion

Alle Grafikobjekte aus Arbeitsmappe entfernen

```
Dim OBJ As ShapeRange
Dim i As Integer
For i = 1 To Sheets.Count
  Sheets(i).Activate
  With Sheets(i)
    If .Shapes.Count > 0 Then
      .Shapes.SelectAll
      Selection.Delete
    End If
  End With
Next i
```

Lesen Sie das ”=”-Zeichen nicht als ”ist gleich”, sondern als ”erhält den Wert von”.