

Inhaltsverzeichnis

1.	Etwas Theorie	19
1.1	Es begann mit BASIC	19
1.2	Vom Kalkulierer zum Programmierer	20
1.3	Der persönliche Anspruch	22
1.3.1	Und was ist mit Google?	22
1.4	Strukturiert programmieren	24
1.4.1	Das Pflichtenheft	24
1.4.2	EVA	24
1.4.3	Datenflussplan	25
1.5	Professionell programmieren	26
1.5.1	Zehn goldene Regeln für professionelle VBA-Makros	27
1.5.2	Das Tao der Programmierung	29
1.6	Los geht's – Vorbereitung und erste Makros	30
1.6.1	Entwicklertools bereitstellen	30
1.6.2	VBA-Editor starten, wechseln zwischen den Fenstern	31
2.	Der VBA-Editor	32
2.1	Die wichtigsten Voreinstellungen im VBA-Editor	32
2.1.1	Menüs und Symbolleisten	32
2.1.2	Unterenfenster	34
2.1.3	Optionen	34
2.2	Module	35
2.2.1	Modul exportieren und importieren	35
2.2.2	Modul löschen	36
2.3	Das erste Makro	36
2.3.1	Hallo Welt	37
2.3.2	Makromappe gleich speichern	38
2.3.3	Kommentare	39
2.3.4	Ein paar Makros zum Üben	39

3. Objekte, Eigenschaften und Methoden 43

3.1	Alles ist Objekt.....	43
3.2	Collections	44
3.2.1	Objektreferenzen vereinfachen	46
3.2.2	With und End With.....	46
3.2.3	Objekteigenschaften und Methoden	47
3.2.4	Ereignisse	48
3.2.5	Das Objektmodell »Autohaus«	48
3.3	Der Objektkatalog.....	49
3.4	Objektsuche und Objekthilfe.....	51
3.5	Objektbibliotheken und Verweise	52
3.6	Klassen.....	54
3.6.1	Klassenmodul anlegen.....	55
3.6.2	Klasse anwenden.....	55

4. Variablen und Konstanten 57

4.1	DIM deklariert Variablen	57
4.1.1	Option Explicit – Variablendeklaration erforderlich	57
4.2	Regeln für die Variablenbenennung	58
4.3	Gültigkeit von Variablen	59
4.4	Datentypen in Variablen	60
4.5	Objektvariablen	62
4.6	Konstanten.....	64
4.7	Datenfelder.....	65
4.8	Namenskonventionen für Variablen und Konstanten	66

5. Makros schreiben und testen..... 69

5.1	Wozu der Unterstrich?	69
5.2	Camel Case.....	70
5.3	Schrittweise testen	71

5.4	Schrittmodus per Stop-Anweisung	72
5.5	Unterprogramme testen.....	73
5.6	Haltepunkte.....	73
5.7	Das Direktfenster	74
5.8	Ausgabe in das Direktfenster per Makro	75
5.9	Das Lokal-Fenster	76
5.10	Das Überwachungsfenster.....	76
5.11	Makros kompilieren.....	78
5.12	Makro abbrechen	79

6. Der Makrorekorder..... 81

6.1	Vor- und Nachteile des Makrorekorders	81
6.2	Mit dem Rekorder aufzeichnen.....	82
6.2.1	Neuer Ordner, neues Modul.....	83
6.2.2	Aufgezeichnetes Makro ausführen.....	84
6.2.3	Das Makrorekorder-Symbol in der Statusleiste.....	84
6.2.4	Rekorder starten über das Ansicht-Register.....	85
6.3	Makrobeispiel: SchriftZoomer	85
6.4	Im richtigen Projekt aufzeichnen.....	88
6.4.1	Persönliche Makroarbeitsmappe	89
6.4.2	Neue Arbeitsmappe	89
6.4.3	Diese Arbeitsmappe.....	89
6.5	Modular aufzeichnen.....	89
6.6	Relative Verweise verwenden	91
6.7	Tastenkombination zuweisen	92

7. Kontrollstrukturen 94

7.1	Verzweigungen.....	94
7.1.1	Die If-Anweisung.....	94
7.1.2	Select Case.....	96

7.2	Schleifen	98
7.2.1	Die Zählschleife (For-Next-Schleife)	98
7.2.2	Die For-Each-Schleife	99
7.2.3	Die Do-Schleifen	101
7.2.4	Die While-Schleife	103
7.2.5	Schleifen abbrechen	103

8. Eingaben und Ausgaben..... 106

8.1	Eingabe	106
8.1.1	Eingaben über InputBox.....	106
8.2	Ausgaben	108
8.2.1	MsgBox	108
8.2.2	UserForms.....	120
8.3	Unterprogramme	121

9. Benutzerdefinierte Funktionen 123

9.1	Funktion speichern	123
9.2	Funktion schreiben	123
9.3	Gültigkeit und Namensregelungen für Funktionen	124
9.3.1	Public.....	124
9.3.2	Private.....	124
9.4	Parameterübergabe	124
9.4.1	By Value.....	125
9.4.2	By Reference	125
9.4.3	Optional.....	125
9.5	Funktionsergebnis als Rückgabe	125
9.6	Argument in Klammern	127
9.7	Benutzerdefinierte Tabellenfunktionen	127
9.8	Praxisbeispiele für Funktionen	129
9.8.1	Quersumme berechnen.....	129
9.8.2	Betrag in Euro und Cent aufteilen.....	129
9.8.3	Kalenderwoche berechnen	130

9.9	Funktionen in Prozeduren.....	130
9.9.1	Aufruf einer Funktion aus Prozeduren.....	130
9.10	WorksheetFunctions	133
9.11	Textfunktionen	134
9.11.1	Format	134
9.11.2	InStr.....	135
9.11.3	LCase, UCase	135
9.11.4	Left, Right.....	135
9.11.5	Mid.....	135
9.11.6	Replace.....	136
9.11.7	String	136
9.11.8	Trim	137
9.12	API-Funktionen.....	137
9.12.1	Rechenzeit	138
9.12.2	Anwendung aktiv?.....	139
9.12.3	Fenstertitel.....	139
9.12.4	Bildschirmauflösung	140

10. Dialogprogrammierung mit UserForms 141

10.1	Eine UserForm einfügen	142
10.2	UserForm per Makro starten	144
10.2.1	UserForm modal starten.....	145
10.3	UserForm und Codeblatt	145
10.4	Steuerelemente zeichnen	146
10.4.1	Steuerelemente positionieren und ausrichten	147
10.4.2	Steuerelemente löschen.....	148
10.5	Die Toolsammlung (Werkzeugsammlung).....	149
10.5.1	Zusätzliche Steuerelemente.....	151
10.6	UserForm gestalten	152
10.6.1	UserForm-Knigge – Gestaltungsregeln für Benutzerdialoge.....	153
10.6.2	Checkliste: Allgemeine Regeln für UserForms.....	154
10.6.3	Eigenschaften der UserForm.....	154

10.7	Steuerelemente anlegen, kopieren und positionieren.....	155
10.8	Eigenschaften der Steuerelemente	156
10.9	Steuerelemente benennen.....	157
10.10	UserForm-Ereignisse	158
10.10.1	Initialize	159
10.10.2	QueryClose.....	162
10.10.3	Activate.....	163
10.11	Steuerelemente-Ereignisse.....	163
10.11.1	Schaltflächen.....	164
10.11.2	Beschriftungsfeld.....	170
10.11.3	Textfeld	170
10.11.4	Kombinationsfeld und Listenfeld.....	173
10.11.5	Kontrollkästchen	177
10.11.6	Optionsfeld und Optionsfeldgruppen	179
10.11.7	Umschaltfeld	181
10.11.8	Multiseiten und Register	181
10.11.9	Bildlaufleiste und Drehfeld.....	183
10.11.10	Anzeige.....	183
10.11.11	RefEdit.....	186
10.11.12	Slider (Schiebereglern)	187
10.11.13	Web-Browser	188
10.11.14	TreeView.....	190
10.11.15	ProgressBar	194
10.12	Die Aktivierreihenfolge	195

11. Formularsteuerelemente im Tabellenblatt..... 196

11.1	Formularsteuerelemente.....	197
11.1.1	Formularelemente formatieren	198
11.1.2	Makro zuweisen.....	198
11.2	Schaltflächen.....	199
11.3	ActiveX-Elemente	202
11.3.1	Makrobeispiel: Monate	203

12. VBA im Tabellenblatt 205

12.1	Tabellenblattobjekte	205
12.1.1	Selection und Select – meist überflüssig	206
12.1.2	Range in eckigen Klammern	207
12.1.3	Range.....	207
12.1.4	Cells.....	216
12.1.5	Offset und Resize.....	218
12.1.6	Range-Objekte auslesen.....	221
12.2	Range-Methoden	222
12.2.1	Copy und Paste	222
12.2.2	Clear und Delete.....	223

13. Laufwerke, Ordner und Dateien 225

13.1	CurDir und ChDrive – Verzeichnis und Ordner aktuell	225
13.1.1	Laufwerke auslesen (API).....	227
13.2	Laufwerk und Ordner wechseln.....	228
13.3	Dateien öffnen.....	230
13.3.1	Der Dialog.....	230
13.3.2	Mehrfachselektion auslesen	232
13.3.3	FileDialog	233
13.3.4	Workbooks.Open.....	234
13.3.5	Textdateien.....	236
13.4	Ordner und Dateien verwalten mit dem FileSystemObject	238
13.4.1	Die Existenzprüfungen.....	239
13.4.2	Ordner erstellen, kopieren, löschen	240
13.4.3	Ordner- und Dateinfos auslesen.....	242
13.4.4	Dateispionage mit Windows Shell.....	244
13.5	Dateien speichern	246
13.5.1	Workbook.Save	246
13.5.2	Workbook.SaveAs	246
13.5.3	Speichern mit GetSaveAs.....	247
13.5.4	Textdatei speichern	248
13.6	Dateien löschen	249
13.6.1	Fehler beim Löschen.....	250

14. Ereignisse252

- 14.1 Arbeitsmappenereignisse252
- 14.2 Tabellenblattereignisse254
 - 14.2.1 Worksheet_Activate255
 - 14.2.2 Selection_Change()255

15. Mit Tabellen programmieren259

- 15.1 VBA-Anweisungen für Tabellen.....260
 - 15.1.1 Tabelle anlegen.....261
 - 15.1.2 Prüfen, ob Tabelle existiert.....262
 - 15.1.3 Tabelle in Bereich umwandeln262
 - 15.1.4 Tabelle in Array einlesen.....263

16. VBA und Microsoft Office264

- 16.1 VBA in Microsoft Office – einheitlich und doch völlig anders.....264
- 16.2 EarlyBinding oder LateBinding?.....265
- 16.3 VBA und Outlook.....266
 - 16.3.1 Kontakte via CSV nach Excel exportieren.....267
 - 16.3.2 Kontakte für den Outlook-Import in CSV umwandeln268
 - 16.3.3 Kontakte zwischen Excel und Outlook synchronisieren269
 - 16.3.4 Termine importieren und exportieren273
 - 16.3.5 Outlook-Mails versenden per Makro.....274
 - 16.3.6 Makrobeispiel: Outlook-Mails an Verteiler275
- 16.4 VBA und Word.....279
 - 16.4.1 Der Bibliotheksverweis für Word279
 - 16.4.2 Textmarken in Word befüllen.....280
- 16.5 VBA und Datenbanken284
- 16.6 Die Objektbibliotheken für Datenbankzugriffe284
 - 16.6.1 Access-Daten importieren285
 - 16.6.2 Access-Daten mit DAO auslesen.....286
 - 16.6.3 SQL-Server-Daten mit ADO auslesen289

16.7	SQL im Excel-Makro	290
16.7.1	ODBC	291
16.7.2	ODBC-Abfrage mit MS Query	291
16.7.3	SQL in der Query-Abfrage.....	292
16.7.4	Makrobeispiel: Umsatzanalyse mit SQL.....	293
16.7.5	SQL-Befehle für Excel	295

17. Business Intelligence 296

17.1	VBA im BI	297
17.2	Power Query.....	297
17.2.1	M-Code	298
17.2.2	Power-Query-Abfrage mit dem Makrorekorder aufzeichnen.....	299
17.3	Power Pivot.....	303
17.3.1	Makrobeispiel: Kostentabelle in Power Pivot einbinden	304
17.3.2	Makrobeispiel: PivotTable aus Power-Pivot-Tabelle anlegen	304

18. Menüband und Symbolleiste programmieren 305

18.1	Makros in der Symbolleiste für den Schnellzugriff	305
18.1.1	Makrobeispiel: Environment-Infos.....	305
18.2	Makros im Menüband platzieren.....	306

19. VBA-Makros im Menüband (RibbonX) 307

19.1	XLSX ist ZIP	307
19.2	Der Office RibbonX Editor	308
19.3	Die Office Icon Gallery	308
19.4	RibbonX-Grundlagen.....	309
19.4.1	Office-Namespace	309
19.5	Das Onload-Ereignis.....	311
19.6	Makrobeispiel: VBA-Tools im Menüband	312

20. Makros zertifizieren 315

20.1	Office-Programm für Zertifikate	315
20.2	Zertifikat erstellen	315
20.3	Digitale Signatur erstellen.....	316
20.3.1	Echte Zertifizierung.....	316

21. Know-how von A bis Z 317

21.1	Add-in	317
21.2	AutoFilter	318
21.3	Bildschirm.....	319
21.3.1	Ein- und ausschalten.....	319
21.3.2	Meldungen ein-/ausschalten.....	320
21.3.3	UserForms zoomen.....	320
21.4	Blattschutz	321
21.4.1	Blattschutzpasswort entfernen.....	322
21.5	Datenschnitte	323
21.6	Diagramme	323
21.7	Fehlerbehandlung	325
21.7.1	Fehler verhindern und abfangen	325
21.7.2	On Error.....	325
21.7.3	On Error Goto.....	325
21.8	IntelliSense.....	327
21.9	Makronamen.....	328
21.10	PDF.....	328
21.11	Persönliche Arbeitsmappe.....	328
21.12	PivotTables.....	329
21.13	VBA-Projekt.....	331
21.13.1	Eigenschaften des VBA-Projekts	331
21.13.2	VBA-Projekt schützen	332
21.14	Zeichen und Zeichencodes	333

22. Office-Skripts einsetzen 336

- 22.1 Voraussetzungen.....336
- 22.2 Das Menü »Automatisieren«337
 - 22.2.1 Skripterstellungstools..... 337
 - 22.2.2 Office-Skripts 337
 - 22.2.3 Flowvorlagen 338
- 22.3 Office-Skript aufzeichnen..... 338
- 22.4 Office-Skript bearbeiten339
 - 22.4.1 IntelliSense 340
- 22.5 Office-Skript erstellen341
- 22.6 Grundlagen TypeScript 341
 - 22.6.1 Die Funktion main..... 341
 - 22.6.2 Bereiche 342
 - 22.6.3 Skript mit Schaltfläche ausführen 343
- 22.7 Office-Skripts und Power Automate.....344

Stichwortverzeichnis 346

Makrobeispiel: Alter berechnen

Schreiben Sie ein Makro, das aus einem per Dialog angeforderten Geburtsdatum das Alter berechnet. Da dieses Makro keine ausführbaren Aktionen enthält, müssen Sie die Codezeilen ohne Makrorekorder erstellen.

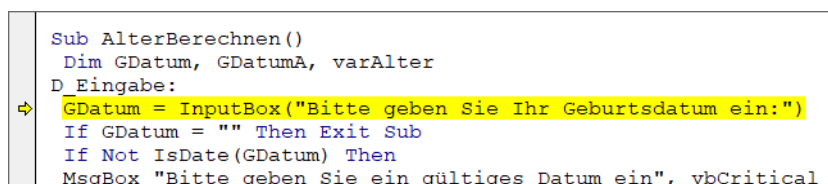
Aktivieren Sie den Visual Basic Editor. Legen Sie ein neues Modul an und schreiben Sie diesen Code:

Makro Nr. 13

```
Sub AlterBerechnen()
    Dim GDatum, GDatumA, varAlter
    D_Eingabe:
    GDatum = InputBox("Bitte geben Sie Ihr Geburtsdatum ein:")
    If GDatum = "" Then Exit Sub
    If Not IsDate(GDatum) Then
        MsgBox "Bitte geben Sie ein gültiges Datum ein", vbCritical
        GoTo D_Eingabe
    End If
    GDatumA = DateSerial(Year(Date), Month(GDatum), Day(GDatum))
    If GDatumA <= Date Then
        alter = Year(Date) - Year(GDatum)
    Else
        alter = Year(Date) - Year(GDatum) - 1
    End If
    MsgBox "Geburtstag: " _
        & Format(GDatum, "dddd, d. MMMM YYYY") & vbCr _
        & "Alter: " & alter
End Sub
```

5.3 Schrittweise testen

Mit dem Schritt-Test wird ein Makro Zeile für Zeile durchgetestet. Sie können Makros schon beim Aufruf im Excel-Fenster schrittweise ablaufen lassen. Markieren Sie das Makro in der Makroliste und klicken Sie auf die Schaltfläche *Schritt*.



```
Sub AlterBerechnen()
    Dim GDatum, GDatumA, varAlter
    D_Eingabe:
    GDatum = InputBox("Bitte geben Sie Ihr Geburtsdatum ein:")
    If GDatum = "" Then Exit Sub
    If Not IsDate(GDatum) Then
        MsgBox "Bitte geben Sie ein gültiges Datum ein", vbCritical
        GoTo D_Eingabe
    End If
    GDatumA = DateSerial(Year(Date), Month(GDatum), Day(GDatum))
    If GDatumA <= Date Then
        alter = Year(Date) - Year(GDatum)
    Else
        alter = Year(Date) - Year(GDatum) - 1
    End If
    MsgBox "Geburtstag: " _
        & Format(GDatum, "dddd, d. MMMM YYYY") & vbCr _
        & "Alter: " & alter
End Sub
```

Abbildung 5.2: Der Schrittmodus: Die gelbe Zeile ist die nächste

Der Visual Basic Editor wird aktiviert, die erste Zeile des Makros erhält einen gelben Hintergrund. Drücken Sie die Taste **F8**, um Zeile für Zeile durch den Code zu wandern. In der Markierungsleiste am linken Rand zeigt ein kleiner Pfeil auf die jeweils nächste Zeile, die zur Ausführung ansteht.

Die Symbolleiste *Debuggen* bietet auch Symbole für das schrittweise Austesten an, der *Einzel-schritt* entspricht der Taste **F8**, mit dem *Prozedurschritt* können Sie so testen, dass Unterprogramme nicht einzeln, sondern als Block durchgerechnet werden.

Mit der Taste **F5** oder einem Klick auf das *Fortsetzen*-Symbol wechseln Sie vom Einzelschritt in die normale Ausführung.

Wenn Sie das Makro ab einer Stelle weiter unten oder oben im Code testen wollen, setzen Sie einfach den Zeiger um. Klicken Sie in die Zeile und drücken Sie **Strg+F9** oder ziehen Sie das Zeigersymbol am linken Rand mit gedrückter Maustaste an die neue Position.

Ziehen Sie den gelben Pfeil am Rand mit dem Mauszeiger an eine neue Position, läuft das Makro ab dieser Zeile weiter. Sie können auch den Cursor in eine andere Zeile setzen und **Strg+F9** drücken.

5.4 Schrittmodus per Stop-Anweisung

Sie können ein Makro so präparieren, dass es an einer bestimmten Codezeile automatisch in den Schrittmodus wechselt. Geben Sie die Anweisung *Stop* ein.


Das Makro durchläuft einen Bereich in der aktiven Tabelle und wechselt in den Schrittmodus, wenn es auf eine Zelle stößt, der das EUR-Zahlenformat zugewiesen ist.




Makro Nr. 14

```
Sub StopTest()
  MsgBox "Makro stoppt in der ersten EUR-Zelle"
  For Each varzelle In Range("F1:F20")
    varzelle.Select
    If InStr(varzelle.NumberFormatLocal, "EUR") = 0 Then
      Stop
    End If
  Next varzelle
End Sub
```

5.5 Unterprogramme testen

Mit der Anweisung *Call* wird ein Unterprogramm aufgerufen. In der Praxis ist es sehr zeitraubend, bereits getestete Routinen im Einzelschritt abzuarbeiten. Verwenden Sie das Symbol *Prozedurschritt* oder diese beiden Tastenkombinationen:

+ überspringt das Unterprogramm, es wird als eine Codezeile ausgeführt.

++ drücken Sie im Unterprogramm, wenn Sie alle weiteren Zeilen ausführen und zum Einzelschrittmodus im aufrufenden Programm zurückkehren möchten.

5.6 Haltepunkte

Haltepunkte sind besonders in längeren Makrocodes von großem Nutzen, wenn die schrittweise Ausführung eines bereits getesteten Codes nicht ausreicht oder nicht mehr nötig ist. Setzen Sie bei den Makroanweisungen, ab denen Excel das Makro schrittweise testen soll, Haltepunkte:


Klicken Sie im Modul auf die Anweisung, bei der Sie den Unterbrechungsmodus einschalten wollen. Setzen Sie mit *Debuggen/Haltepunkt ein/aus* einen Haltepunkt. Die Zeile wird rot unterlegt. Starten Sie das Makro per Klick auf das Symbol *Sub/UserForm ausführen*. Das Makro läuft bis zur Zeile mit dem Haltepunkt und schaltet dort in den Schrittmodus.

```

End If
GDatumA = DateSerial(Year(Date), Month(GDatum), Day(GDatum))
● If GDatumA <= Date Then
    varAlter = Year(Date) - Year(GDatum)
Else
    varAlter = Year(Date) - Year(GDatum) - 1
End If


```

Abbildung 5.3: Ein Haltepunkt

Mit der Taste  springen Sie im Makro von Haltepunkt zu Haltepunkt, setzen Sie diese jeweils an die letzte Zeile eines ausgetesteten Blocks.

Ein Haltepunkt wird genauso gelöscht, wie er gesetzt wurde: Markieren Sie die Anweisung und wählen Sie *Debuggen/Haltepunkt ein/aus*.

Ein schneller Haltepunktschalter

Mit  oder einem Klick auf die Markierungsleiste links wird ein Haltepunkt gesetzt und auch wieder entfernt. *Alle Haltepunkte löschen* aus dem Menü *Debuggen* macht, was es ausdrückt: Die Option entfernt alle gesetzten Haltepunkte in einem Makro.

5.7 Das Direktfenster

Das wichtigste Testwerkzeug ist gerade für Einsteiger das Direktfenster. Es bietet die Möglichkeit, vor oder während des Makrolaufs Objekte, Methoden und Eigenschaften abzurufen, Variablen oder Ausdrücke zu testen oder einfach Zellinhalte, Datei- und Tabellennamen abzufragen.

Öffnen Sie das Fenster mit **(Strg)+(G)** oder *Ansicht/Direktfenster*.

Geben Sie ein Fragezeichen (?) ein. Das ist die Print-Anweisung für dieses Fenster, der Rest der Zeile wird »in das Fenster gedruckt«. Geben Sie eine Anweisung oder einen Teilcode ein und drücken Sie **(↵)**. Das Ergebnis wird unter der Fragezeichenzeile angezeigt, Sie können eine neue Zeile eingeben oder die erste Zeile überschreiben. Fragen Sie im Direktfenster Informationen aus Windows oder Excel ab. Das sind Informationen, die Excel (oder Windows) zur Verfügung stellt und die im Makrocode über Anweisungen, Konstanten oder Objekteigenschaften abgerufen werden:

Systeminfo	Ergebnis
? Date	Tagesdatum
? Time	Aktuelle Uhrzeit
? Application.Username	Name des Excel-Anwenders
? ActiveSheet.Name	Name der aktiven Tabelle
? Application.Version	Versionsnummer von Excel

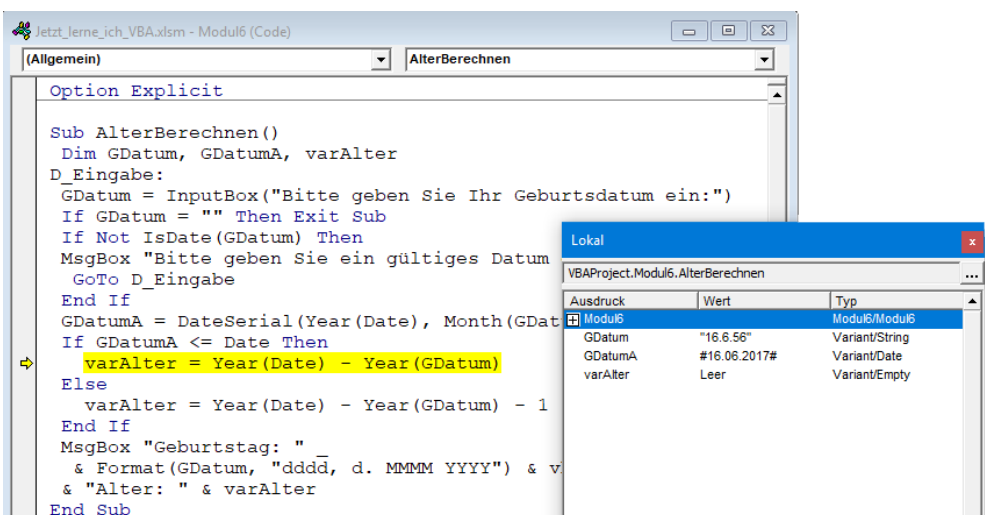


Abbildung 5.4: Das Direktfenster

5.8 Ausgabe in das Direktfenster per Makro

Das Direktfenster trägt den Objektnamen *Debug*, und um es von einer Prozedur aus anzusteuern, verwenden Sie diese Anweisung:

```
Debug.Print xx
```

Sie können alle Abfragen, die Sie eingetippt hatten, auch zu einem Makro zusammenfassen. Schreiben Sie dieses Makro in ein Modul:

Makro Nr. 15

```
Sub XLInfos ()
  Debug.Print Date
  Debug.Print Time
  Debug.Print "Benutzer: " & Application.UserName
  Debug.Print "Programm: " & Application.Name & " " & Application.Version
  Debug.Print "Aktuelle Tabelle: " & ActiveSheet.Name
  Debug.Print "Zellzeiger steht in Zelle: " _
    & ActiveCell.Address
  Debug.Print ActiveWorkbook.Name
End Sub
```

Starten Sie das Makro mit geöffnetem Fenster, sehen Sie, wie die Informationen Zeile für Zeile in das Direktfenster gedruckt werden.

Makrobeispiel: Makro nach Bedingung anhalten

Sie können das Debug-Objekt auch dazu benutzen, das Makro anzuhalten, wenn eine bestimmte Bedingung nicht erfüllt ist. Verwenden Sie die Methode *Assert*, die das Makro anhält. Hier ein Beispiel: Das Makro durchläuft in einer Schleife die ersten 50 Zellen der Spalte A, stoppt bei der ersten Zelle, die leer ist, und wechselt in den Einzelschritt.

Makro Nr. 16

```
Sub DebugAssertTest()
  For i = 1 To 50
    Cells(i, 1).Select
    Debug.Assert (ActiveCell.Value <> "")
  Next i
End Sub
```


5.9 Das Lokal-Fenster

Im Lokal-Fenster finden Sie alle deklarierten Variablen der aktiven Prozedur oder Funktion, wenn Sie im Einzelschrittmodus arbeiten. Aktivieren Sie dieses Fenster über *Ansicht/Lokal-Fenster*.

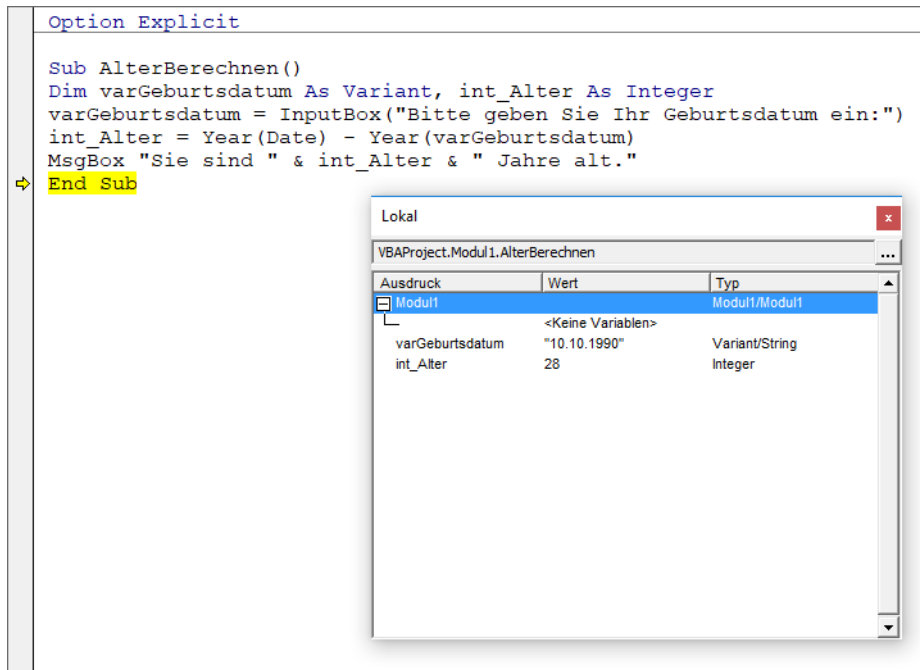


Abbildung 5.5: Im Lokalfenster sehen Sie alle Variablen

Sie können den Variableninhalt in der zweiten Spalte ändern, setzen Sie einfach den Cursor in den Eintrag und geben Sie einen neuen Wert ein. Das Symbol mit den drei Punkten rechts oben in der Ecke bietet alle Funktionen und Prozeduren des aktiven Moduls an.

5.10 Das Überwachungsfenster

Um die Änderungen in Variablen und anderen beweglichen Teilen des Makros gezielt kontrollieren zu können, schalten Sie das Überwachungsfenster ein. Sie können das Fenster einfach einblenden und Teile des Makrocodes darin einfügen:

Wählen Sie *Ansicht/Überwachungsfenster*. Markieren Sie einen Teil des Makrocodes, eine Variable, ein Objekt oder einen Ausdruck. Wählen Sie *Debuggen/Überwachung hinzufügen*, geben Sie die Überwachungsart an und bestätigen sie, um die Über-

wachung einzufügen. Für eine schnelle Überwachung ziehen Sie den markierten Codeteil einfach mit gedrückter Maustaste in das Fenster. Die Überwachungsart kontrollieren Sie mit *Überwachung bearbeiten* aus dem Kontextmenü der rechten Maustaste.

Testen Sie das Überwachungsfenster mit einem Makro, das Zufallszahlen generiert:

Makro Nr. 17

```
Sub Zufallszahl()
  Dim intZähler
  For intZähler = 1 To 10
    Randomize
    TestWert = (Int(10 * Rnd) + 1)
  Next intZähler
End Sub
```

Ziehen Sie die Variablen *intZähler* und *TestWert* in das Überwachungsfenster und testen Sie das Makro mit **F8**. Mit jedem Durchlauf der Schleife erhalten die beiden Variablen auf der Überwachungskarte andere Werte. Der Schleifenzähler zählt sequenziell von 1 bis 10, die Zufallszahl erhält Zufallswerte zwischen 1 und 10.

Eine weitere Variante der Überwachung ist die Unterbrechung: Sie können eine Überwachung so definieren, dass sie die Ausführung eines Makros unterbricht und in das Textfenster schaltet, sobald eine Variable einen bestimmten Wert angenommen oder sich verändert hat. Fügen Sie für unser Beispiel eine Überwachung ein, die das Makro unterbricht, wenn eine Zufallszahl erzeugt wird, die größer als 90 ist.

Wählen Sie *Debuggen/Überwachung hinzufügen*. Geben Sie diesen Ausdruck ein:

```
TestWert 5
Unterbrechen, wenn der Wert True ist
```

Wählen Sie die Überwachungsart. Starten Sie das Makro mit **F5**. Der Editor stoppt die Ausführung und blendet das Testfenster ein, sobald eine Zufallszahl größer als fünf erzeugt wurde.

Art der Überwachung legt fest, wie VBA auf den Überwachungsausdruck reagiert. Wählen Sie *Überwachungsausdruck*, wird dieser zusammen mit seinem aktuellen Wert im Überwachungsfenster angezeigt. Im Haltemodus wird der Wert automatisch aktualisiert. Wählen Sie *Unterbrechen, wenn der Wert True ist*, wechselt das Makro automatisch in den Haltemodus, wenn der Ausdruck als wahr oder als beliebiger Wert ungleich null ausgewertet wird (nicht in Zeichenfolgenausdrücken). Mit *Unterbrechen, wenn Wert geändert wurde* wechselt das Makro automatisch in den Haltemodus, wenn sich der Wert des Ausdrucks innerhalb des angegebenen Kontextes ändert.

Um eine Überwachung wieder zu löschen, markieren Sie sie im Überwachungsfenster und drücken die Taste **(Entf)**. Ein Doppelklick auf die Zeile öffnet die Dialogbox zur Bearbeitung der Überwachung (auch unter *Extras/Überwachung bearbeiten*).

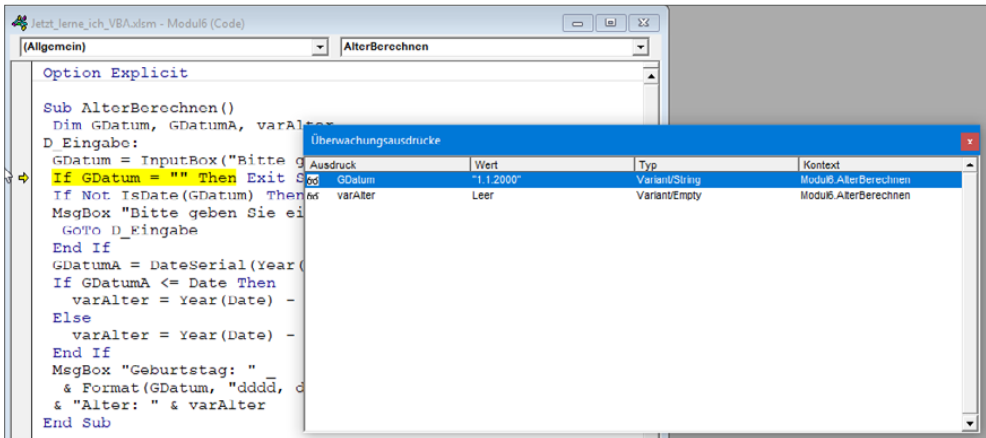


Abbildung 5.6: Das Überwachungsfenster

5.11 Makros kompilieren

Mit der Kompilierung eines Makros werden alle Makros noch mal gründlich auf Fehler getestet:

1. Markieren Sie das Projekt, das Sie kompilieren wollen.
2. Wählen Sie *Debuggen/Kompilieren von Projektxx*.

Im Unterschied zur Kompilierung in selbstständigen Programmiersprachen wie Visual Basic, C++ oder Java findet keine Umsetzung in Maschinensprache statt (nur intern), es wird auch keine ausführbare Datei erzeugt. Der Compiler endet auch ohne weitere Kommentare, wenn keine Fehler mehr im Code entdeckt wurden, und der Befehl im *Debuggen*-Menü ist inaktiv, solange das Projekt nach der Kompilierung nicht verändert wurde.

Mit der *bedingten Kompilierung* haben Sie die Möglichkeit, zwei Versionen von Makrocodes zu schreiben, eine *Testversion* und eine *Auslieferversion*. Wenn Sie Anweisungen wie *Stop*, *Debug.Assert* oder *Debug.Print* oder Meldungen im Makro haben, die nur beim Austesten zur Laufzeit auftauchen sollen und nicht in der Endversion, die an Kunden ausgeliefert wird, verpacken Sie diese in bedingte Kompilierungsblöcke. Sie brauchen zunächst eine Compilervariable, geben Sie diese mit einer *#CONST*-Anwei-

sung am Kopf des Moduls ein. Hier heißt die Variable *myTest*, sie wird auf den Boolean-Wert *True* gesetzt:

```
#Const myTest = True
```

Schreiben Sie ein Makro, das eine Reihe von Zahlen in einem Zellbereich abgreift und in eine Variable aufsummiert.

Makro Nr. 18

```
Sub ZahlenCheck()  
  Dim varZelle, lngSumme  
  Range("Testzahlen").Select  
  For Each varZelle In Selection  
    lngSumme = lngSumme + varZelle  
  Next varZelle
```

Die Meldung der Summe erfolgt über eine bedingte Kompilierung, dazu wird vor *If* und *End If* jeweils ein #-Zeichen gesetzt.

```
#If myTest = True Then  
  MsgBox "Summe: " & lngSumme  
#End If  
End Sub
```

Dieser Block wird nur ausgeführt, wenn die Compilervariable auf *True* gesetzt ist. Sie können jetzt in allen Makros Anweisungsblöcke schreiben, die vom Wert der Compilervariablen abhängig sind. Ist das Makro fertig getestet, setzen Sie die Variable auf *False* und die Blöcke werden nicht mehr ausgeführt. Schneller geht's mit einem Unterprogramm. Verpacken Sie die kompilierten Bedingungen in eine Subroutine und rufen Sie diese auf:

```
Call Check  
Sub Check  
#If . . .  
#End If  
End Sub
```

5.12 Makro abbrechen

Auch das wird Ihnen passieren: Eine Schleife läuft endlos, weil das Schleifenende nicht zu finden ist, oder ein Datenzugriff dauert ewig, weil Excel auf den Server wartet. Um ein laufendes Makro abzubrechen, drücken Sie einfach die Tastenkombination **Strg+Pause**.